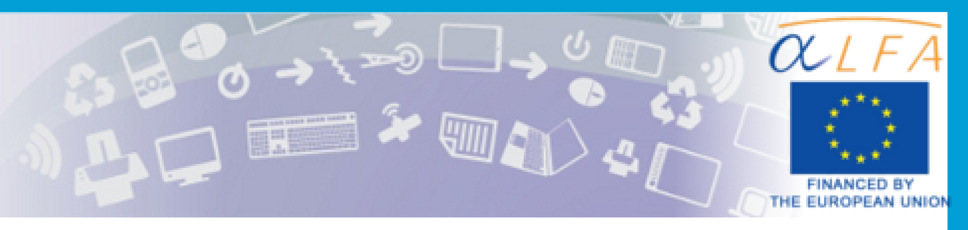
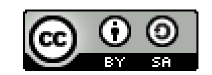




Elementos esenciales para programación: Algoritmos y Estructuras de Datos



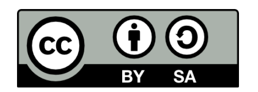
AUTORES

Gracia María Gagliano
Cristina I. Alarcón
Laura M. Angelone
Edison Isaías Del Rosario Camposano
Pedro Cardona
Fernando Guspi
José Eder Guzmán Mendoza
Zenón Luna
Pablo Augusto Magé
Jaime Muñoz Arteaga

Elementos escenciales para programación: Algoritmos y Estructuras de Datos

1a ed. - Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn), 2014. 202 pag.

Primera Edición: Marzo 2014
Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn)
http://www.proyectolatin.org/









Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.
El Proyecto LATIn está conformado por: Escuela Superior Politécnica del Litoral, Ecuador (ESPOL); Universidad Autónoma de Aguascalientes, México (UAA), Universidad Católica de San Pablo, Perú (UCSP); Universidade Presbiteriana Mackenzie, Brasil(UPM); Universidad de la República, Uruguay (UdelaR); Universidad Nacional de Rosario, Argentina(UR); Universidad Central de Venezuela, Venezuela (UCV), Universidad Austral de Chile, Chile (UACH), Universidad del Cauca, Colombia (UNICAUCA), Katholieke Universiteit Leuven, Bélgica (KUL), Universidad de Alcalá, España (UAH), Université Paul Sabatier, Francia (UPS).

Índice general

0.1 Prólogo

Este libro está principalmente dirigido a estudiantes universitarios que se inician en las carreras de Ingeniería. Son los que deben interiorizarse en el manejo de ciertos recursos propios de la disciplina sin orientarse a ningún lenguaje en particular. La finalidad de este enfoque es ampliarles la perspectiva para que en su vida profesional no sólo puedan implementar programas en los lenguajes clásicos de programación sino que también sean capaces de manejar las nuevas herramientas de software que, en muchos casos, permiten optimizar su rendimiento mediante el desarrollo de macroinstrucciones especiales.Es así que en la elaboración del libro hemos decidido tratar los conceptos de manera no muy extensa y, en la forma más sencilla posible. No sólo para que se constituya en un soporte de los contenidos a desarrollar por los docentes sino para que les permita a los estudiantes reflexionar sobre la forma de llegar a un algoritmo como solución a un problema.A los fines didácticos, representamos los algoritmos en algunos casos con pseudocódigo y en otros, con diagrama de flujo o con ambos tipos de representaciones.

El libro está constituido por nueve capítulos. Se inicia con una introducción a los recursos de la algoritmia desde los conceptos básicos, pasando por la organizacion de las acciones según la técnica de programación estructurada y la implementación de subalgoritmos. Se incluye también el concepto y uso apropiado de estructuras de datos como arreglos, registros y cadenas, como así también el de archivo de datos y su tratamiento. En un capítulo aparte desarrollamos algunos procesos específicos para el ordenamiento e intercalación de conjuntos de valores y para la búsqueda en los mismos,Presentamos la temática referida a las Estructuras de Selección y a las Estructuras de Iteración en tres capítulos cuyos contenidos se complementan, pues fueron abordados desde distintos enfoques (teórico, práctico y teórico-práctico) a partir de la visión de diferentes autores.El libro culmina con una descripción de la representación interna de la información numérica y de carácter,nociones imprescindibles para la comprensión del funcionamiento interno de una computadora.

Agradecemos a los Profesores de la Facultad de Ciencias Exactas, Ingeniería y Agrimensura de la Universidad Nacional de Rosario, Ingeniero Zenón Luna e Ingeniera María Alicia Morelli por su contribución con valiosas sugerencias y consejos para mejorar el contenido del libro. El primero ha participado en la producción del capítulo **Estructura de datos: Archivos** y la segunda, en la revisión de los capítulos **Introducción a la Informática, Técnicas de programación. Organización de las acciones, Subalgoritmos, Estructuras de datos, Ordenamiento, búsqueda e intercalación, Estructuras de datos: Archivos.**Hacemos lo propio con los profesores de la Universidad Autónoma de Aguascaliente Pedro Cardona S. y Jaime Muñoz Arteaga que han colaborado en la producción del capítulo **Descripción de las estructuras selectivas y repetitivas**.

1 — Introducción a la Informática

Laura M. Angelone

Es una pérdida de tiempo que personas cualificadas pierdan horas como esclavos en la labor de calcular, lo que podría ser delegado en cualquier otro si se pudieran utilizar máquinas.

Gottfried Wilhelm Leibniz (1646-1716)

1.0.1 Qué es la Informática?

La Informática nace bajo la idea de ayudar al hombre en aquellos cálculos rutinarios, donde frecuentemente existe una repetición de tareas. Ejemplo de ello es la gestión de un censo, tal es el caso del primero automatizado en 1890 en EEUU utilizando tarjetas perforadas. En esa época se pensó que una máquina no sufriría cansancio ni cometería errores. Con el tiempo esta idea se fue afianzando en la sociedad, y es hoy día que se sigue trabajando para mejorar las prestaciones de los sistemas de cómputo.

Si repasamos la historia veremos que la Informática estuvo entre nosotros desde tiempos lejanos. A continuación transcribimos algunas ideas de importantes pensadores relacionados con la Informática. Para ubicarse en la historia, debe notarse que los primeros modelos de computadoras aparecieron alrededor del año 1940.

Gottfried Wilhelm Leibniz (1646-1716) fue uno de los grandes pensadores de los siglos XVII y XVIII, decía que "*Es una pérdida de tiempo que personas cualificadas pierdan horas como esclavos en la labor de calcular, lo que podría ser delegado en cualquier otro si se pudieran utilizar máquinas*". Basándose en los trabajos de Pascal construyó una calculadora compuesta por cilindros dentados con longitud incremental que podía sumar, restar, multiplicar y dividir automáticamente, conocida como la rueda de Leibniz. Algunas máquinas basadas en estos mismos principios han jugado un papel preponderante en la Segunda Guerra Mundial, e incluso algunas están todavía en uso. Leibniz fue también el primer pensador occidental en investigar la aritmética binaria y sentó las bases de la lógica simbólica, basada en símbolos y variables abstrayendo la semántica de las proposiciones.

Ada Byron (1815-1852) desarrolló los primeros programas para la Máquina Analítica de Babbage (1833), debido a estos trabajos, se la considera el primer programador de computadoras del mundo. Ella escribió "*La Máquina Analítica no tiene la pretensión de crear nada. Puede realizar cualquier cosa siempre que conozcamos cómo llevarla a cabo. Puede seguir análisis; pero es incapaz de descubrir relaciones analíticas o verdades. Su potencialidad es la de ayudarnos a hacer posible aquello sobre lo que tenemos un conocimiento previo*. Muchas han sido las

mujeres que han realizado grandes aportes a la Informática, aún así Ada Byron es la única mujer que cuenta con un lenguaje de programación que lleva su nombre: en 1979 el Departamento de Defensa de los Estados Unidos creó un lenguaje de programación basado en Pascal que se llamó lenguaje de programación Ada en su honor.

Herman Hollerith (1860-1929) fue un estadístico estadounidense que inventó la máquina tabuladora. Es considerado como el primer informático, es decir, el primero que logra el tratamiento automático de la información (Informática = Información + automática). En esos tiempos, los censos se realizaban de forma manual, con el retraso de unos 10 años en su procesamiento. Ante esta situación, Hollerith comenzó a trabajar en el diseño de una máquina tabuladora o censadora, basada en tarjetas perforadas que patentó en el año 1889. Un año después incluyó la operación de sumar con el fin de utilizarla en procesos de contabilidad.

Alan Turing (1912-1954) fue un matemático, lógico, científico de la computación, criptógrafo y filósofo británico. Es considerado uno de los padres de la ciencia de la computación siendo el precursor de la informática moderna. Proporcionó una influyente formalización de los conceptos de algoritmo y computación, la famosa máquina de Turing. Durante la Segunda Guerra Mundial, trabajó en descifrar los códigos nazis, particularmente los de la máquina Enigma. Tras la guerra diseñó una de las primeras computadoras electrónicas programables digitales en el Laboratorio Nacional de Física del Reino Unido (1945). Entre otras muchas cosas, también contribuyó de forma particular e incluso provocativa al enigma de si las máquinas pueden pensar, es decir a la Inteligencia Artificial. Turing decia: *"Las máquinas me sorprenden con mucha frecuencia."*

John Von Neumann (1903-1957) fue un matemático húngaro que realizó contribuciones fundamentales en física cuántica, análisis funcional, teoría de conjuntos, ciencias de la computación, economía, análisis numérico, cibernética, hidrodinámica, estadística y muchos otros campos. Está considerado como uno de los más importantes matemáticos de la historia moderna. Diseñó una arquitectura de computadoras que lleva su nombre, y aún es utilizada en casi todas las computadoras personales, microcomputadoras, minicomputadoras y supercomputadoras. Von Neumann decía "*Podría parecer que hemos llegado al límite de lo que es posible lograr con la tecnología informática, aunque hay que tener cuidado con tales declaraciones, ya que tienden a sonar bastante tontas en cinco años."*

El término *Informática* nace recién en la década de 1960 en Francia bajo la denominación INFORMATIQUE, debido a la contracción de las palabras INFORmation y autoMATIQUE, es decir el tratamiento de la información por medios automáticos. En las múltiples publicaciones, Informática se define de diversas maneras pero siempre ronda la misma idea: ***el tratamiento automático de la información***.

A continuación veremos algunas definiciones.

- **INFORMÁTICA** es la ciencia que estudia el tratamiento automático y racional de la información. Se habla de tratamiento automático debido a que son máquinas las que procesan la información y se dice racional por estar los procesos definidos a través de programas que siguen el razonamiento humano.
- **INFORMÁTICA** es el estudio de los algoritmos y de las computadoras - de sus teorías, sus modelos abstractos, su realización mecánica, su fiabilidad y verificación, su medida y eficacia, su descripción linguística y su contexto social.
- Las **Ciencias de la Computación o Informática** como se le llama en algunos países hispanoamericanos, es la disciplina que busca establecer una base científica para diversos

temas, como el diseño de computadoras, la programación de computadoras, el proceso de información, la elaboración de algoritmos para resolver problemas y el proceso algorítmico en sí.

1.1 Del mundo real a la solución por computadora

1.1.1 Informatizar la resolución de un problema

El proceso de informatizar la resolución de problemas del mundo real, tales como resolver una ecuación matemática o realizar la liquidación de sueldos de los empleados de una empresa, conlleva una serie de pasos que debemos seguir para obtener una respuesta encuadrada a los datos provistos. En la Figura 1 se muestra tal proceso.

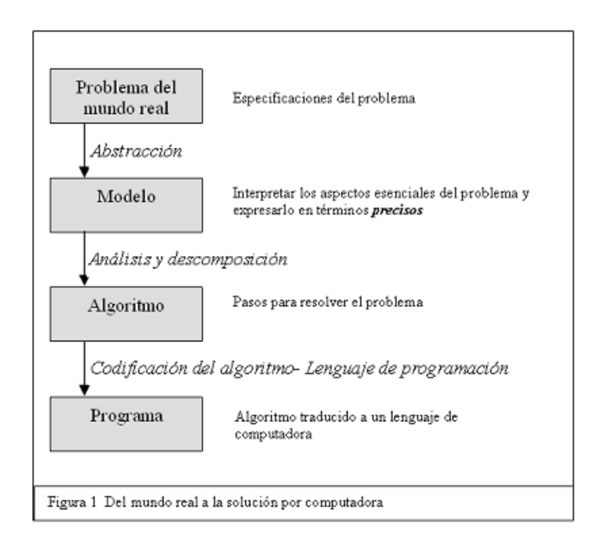


Figura 1 Del mundo real a la solución por computadora

Como primer paso se deberá conocer las especificaciones del ***problema***, es decir, analizar y determinar en forma clara y concreta el objetivo que se desea. Analizar los datos que se disponen y cuales son los resultados que se desean. Una vez que se conoce el problema en detalle, se puede intentar realizar un ***modelo*** del mismo, es decir, abstraer el problema tratando de encontrar los aspectos principales que se pueden resolver, los datos que se han de procesar y el contexto del problema, simplificando su expresión.Disponiendo de un panorama más claro del problema, sus datos y resultados, se puede escribir una serie de acciones que seguidas paso a paso resuelvan el problema. Esto es el ***algoritmo***. Si esta etapa es exitosa, se traduce el algoritmo, mediante un lenguaje de programación, para convertirlo en un ***programa*** que podrá ser interpretado por la computadora para su ejecución solución en forma rápida y eficaz.

Un algoritmo es una forma de describir la solución de un problema, explicando paso a paso como se debe proceder para llegar a una respuesta encuadrada a los datos disponibles, en un tiempo finito

1.1.2 Resolución de problemas con computadora

Existen cinco etapas a tener en cuenta para resolver un problema que luego será ejecutado por una computadora en forma rápida y eficaz, a saber:

1. Análisis del problema, se formula y analiza el problema en su contexto del mundo real. COMPRENDER EL PROBLEMA.
2. Diseño de una solución, se elige una metodología para hallar la solución del problema. TRAZAR UN PLAN PARA RESOLVERLO.

3. Escritura de un algoritmo, se expresa la metodología del punto anterior de forma tal que pueda ser interpretado por el procesador que lo va a ejecutar. ESCRIBIR EL ALGORITMO.
4. Codificación del algoritmo. Un algoritmo es una especificación simbólica que debe traducirse a un programa a partir del conocimiento de un lenguaje de programación para poder ser ejecutado por la computadora. ESCRIBIR EL PROGRAMA O CODIFICAR EL ALGORITMO.
5. Ejecución y validación del algoritmo-programa. La etapa siguiente a escribir el programa, es la verificación de funcionamiento del mismo, es decir, ver si el programa conduce al resultado deseado con los datos dados del mundo real. PROBAR EL PROGRAMA.

La primera etapa está dada por entender el enunciado del problema. Es importante que se conozca lo que se desea que realice la computadora, mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.
Cuando se ha comprendido lo que se desea de la computadora, es necesario hacer un ***Análisis del problema***, definiendo:

- Los ***datos*** de entrada.
- Cual es la información que se desea producir, los ***resultados***.
- Los métodos y fórmulas que se necesitan para procesar los datos para arribar a los resultados, la ***metodología de resolución.***

Una vez que se tiene en claro la metodología de resolución, se escribe el algoritmo en forma simbólica. En una primera etapa, es aconsejable probar que el algoritmo propuesto realmente resuelva el problema planteado, utilizando papel, lápiz y nuestra mente.

En una segunda etapa, este algoritmo podrá ser traducido en un lenguaje reconocible por la computadora, generando así el **programa**.La serie de instrucciones del programa se la conoce como código fuente, el cual se escribe en un lenguaje de programación, generalmente un lenguaje de alto nivel (comprensible por el hombre, como Pascal, Fortran o C, entre otros).
El programa podrá ser probado en la computadora. En este caso, pueden aparecer errores de sintaxis y/o de semántica
Los **errores de sintaxis** se refieren a que alguna instrucción está mal escrita y por lo tanto el procesador no puede reconocerla. Son simples de detectar y de modificar. Esta operación la resuelve el traductor del entorno de programación, el Compilador o el Intérprete. Si en el proceso de traducción se detectan errores de sintaxis, habrá que volver al punto 4) para codificar correctamente.
Los **errores de semántica** se refieren a cuestiones de la lógica de la solución, y son los más complicados de hallar. Es en el proceso de Validación donde se detectan los errores de lógica. Habrá que volver al punto 1) para interpretar correctamente el enunciado del problema para rever y modificar el algoritmo propuesto o proponer una nueva solución al problema dado.
Algo muy importante a tener en cuenta cuando se escriben programas es la **Documentación** del mismo. Esto se refiere a los comentarios que se añaden al código fuente para hacer más claro el entendimiento del programa: descripción del funcionamiento del programa, descripción del problema, nombre del autor, entre otros. A menudo un programa escrito por una persona, es usado por otra, o por ella misma después de un tiempo. Por ello la documentación sirve para ayudar a comprender la lógica del programa, para re-usarlo o para facilitar futuras modificaciones (proceso de **mantenimiento**).

1.1.3 Dar el primer paso

En principio, cualquier algoritmo que diseñemos para ser ejecutado en una computadora, puede ser realizado a mano, con papel y lápiz, suponiendo que nosotros simulemos ser el

procesador y que disponemos del tiempo suficiente para llevarlo a cabo. En realidad, encargamos a la computadora la ejecución de los pasos que componen un algoritmo porque es capaz de completarlo en un tiempo mucho menor del que nosotros emplearíamos, y porque la computadora es menos proclive a cometer errores que nosotros, quienes por distracción o cansancio erramos habitualmente.

Sin embargo, las computadoras sólo pueden ejecutar algoritmos que se componen de acciones que puede entender. Por lo tanto, es necesario conocer bien cuáles son dichas acciones para diseñar algoritmos bien escritos.

Saber cuál es el algoritmo que resuelve un problema dado, suele ser un paso difícil para todo aquel que no tenga práctica en el tema. Para escribir algoritmos, el primer paso necesario es plantear cuál sería la solución del problema en términos comunes, en palabras habituales, haciendo uso de todo lo que esté a nuestro alcance, la imaginación, la matemática, el dibujo, el intercambio de ideas, la lógica.

No existe una técnica establecida para encarar la solución de un problema. La íntima asociación entre el proceso de descubrimiento de algoritmos y el de resolución de problemas, en general ha hecho que los científicos de la computación se unan a otras disciplinas en la búsqueda de mejores técnicas para resolverlos. Pero la capacidad para resolver problemas sigue siendo más una aptitud artística que debe desarrollarse individualmente, que una ciencia precisa por aprender.

1.1.4 El modelo computacional

Para encarar la resolución de un problema por computadora hay que considerar como punto de partida que el procesamiento de la información lleva en sí el hecho de tomar datos, elaborarlos y emitir resultados de acuerdo a dichos datos y a la lógica del proceso que se está ejecutando en ese momento. Por lo tanto, se puede pensar en un modelo computacional compuesto de tres partes: ENTRADA, PROCESO y SALIDA, como se muestra en la Figura 2.

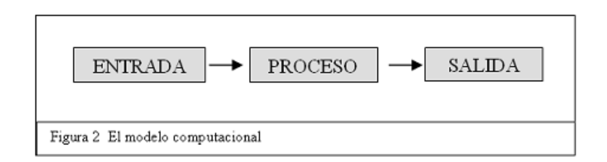


Figura 2 El modelo computacional

Una vez entendido el problema hay que desentrañar del enunciado los datos necesarios para comenzar el proceso de resolución, abstrayéndose del proceso de resolución. Habría que preguntarse "¿ Qué necesito para resolver el problema, si yo fuese el procesador?" "¿ Qué datos necesito que me de otra persona para poder obtener la solución del problema?", y anotarlos como ***ENTRADA* o DATOS**.Una vez determinados los datos, se deberá determinar cuál es el/los resultado/s, es decir, qué se obtiene con los datos de entrada y anotarlos como ***SALIDA* o RESULTADOS.**

A veces pueden ser más fáciles de visualizar los resultados antes que los datos de entrada, en dicho caso habría que preguntarse "¿Para obtener este resultado, qué información necesito?".

Luego se preguntará "¿ Cuáles son los pasos para llegar al resultado partiendo de los datos?", con lo cual se podrá escribir el ***PROCESO* o ALGORITMO** asociado.

Por ejemplo, si se desea realizar un algoritmo para calcular el área y el perímetro de un círculo en función de su radio, las diferentes partes del modelo computacional serán:

ENTRADA:	radio
PROCESO:	Area= π radio^2 Perímetro= 2 π radio
SALIDA:	Area y Perímetro

El proceso (algoritmo) asociado a este problema será:

Dar el valor del radio y guardar ese valor con el nombre R
Calcular el área como pi * R * R
Calcular el perímetro como 2 * pi * R
Informar los resultados de área y perímetro

Sin embargo, el proceso (algoritmo) expresado en forma coloquial no puede ser entendido por la computadora. Por lo cual habrá que escribirlo siguiendo reglas especiales que la computadora entienda.
Vale la pena observar que la relación entre entrada, proceso y salida no es algo que necesariamente se realice en este orden: primero ingresar todos los datos, luego procesarlos, y por último exhibir todos los resultados. P. ej. un algoritmo puede solicitar la entrada de algunos datos, obtener ciertos resultados, y en base a esto pedir nuevos datos y mostrar otros resultados sobre la marcha del proceso.

1.1.5 Datos e información

Por último, describiremos la diferencia que existe entre Datos e Información, términos que se pueden pensar como sinónimos.
En el ambiente de la Informática, el término información puede aplicarse a casi cualquier cosa que pueda comunicarse, tenga o no valor.

Los **datos**, en principio, son información no elaborada, que una vez procesados (comparados, ordenados, sumados, etc.) constituyen información útil.

Según esto, las palabras, los números, las imágenes de este libro son símbolos que representan datos. Si subrayase una frase, se añadirá información a la página.

En una dada situación problemática se puede disponer de muchos datos, de los cuales sólo algunos pueden ser útiles para la resolución del problema, es decir, considerados información útil.

Entonces podemos decir que la **información** es el *conocimiento producido como resultado del procesamiento de los datos*.

Otro término que se utiliza es *Tratamiento de la información,* referido al conjunto de operaciones que se realizan sobre una dada información. Esto es “lectura de datos”, “preservar datos”, “comparar datos”, “procesar aritméticamente datos”, “presentar resultados”. Una adecuada combinación de estas operaciones lleva a resolver los problemas.En el subcapítulo siguiente comenzaremos a formalizar lo vertido en esta sección.

1.2 Algoritmos

En el presente capítulo desarrollaremos el concepto de algoritmo, base de la temática de este libro.

1.2.1 Qué es un algoritmo?

El término **algoritmo** deriva del nombre del matemático persa Abu Ja’far Mohammed ibn Musa Al-Khwarizm (Mohammed, padre de Ja’far, hijo de Moisés, nacido en Khowarizm), el cual vivió alrededor del año 825 después de Cristo. Al-Khowarizm escribió tratados de Aritmética y

Álgebra. La acepción original fue *algorism* y hacía referencia al proceso de prueba de cálculos realizados utilizando números arábigos, tema central del libro de Al-Khowarizm.

> *Un* **algoritmo** *es una forma de describir la solución de un problema, explicando paso a paso como se debe proceder para llegar a una respuesta encuadrada a los datos disponibles, en un tiempo finito.*

Seguramente existirán distintos algoritmos para resolver un mismo problema y todos serán correctos, cada uno de ellos pensado por una persona diferente, cada uno de ellos con un diseño distinto, cada uno de ellos con un sello propio. Escribir algoritmos es un arte, en el que cada persona le dará su estilo personal. Las características de un algoritmo son:

- Un algoritmo debe ser **preciso**, es decir, debe indicar claramente, sin ambigüedades, cada uno de los pasos a seguir para conseguir el objetivo propuesto.
- Un algoritmo debe estar **exacto**, es decir, que si se sigue el algoritmo varias veces con el mismo juego de datos, los resultados obtenidos deben ser los mismos.
- Un algoritmo debe ser **finito**, de tiempo finito, su ejecución debe concluir en algún momento.

¿Quién "ejecuta" estas acciones? La respuesta es: un procesador.

> *Un* **procesador** *es aquel sujeto o máquina que puede entender un enunciado y ejecutar el trabajo indicado en el mismo.*

La sintaxis y semántica a utilizar para escribir un algoritmo depende del lenguaje que conoce el procesador, pues según cuál sea éste, habrá que describir las tareas con más o menos detalle.

Entonces, frente a un problema, sabiendo quien será el procesador y su lenguaje de comunicación, podremos describir la solución del problema mediante un algoritmo que pueda ser entendido y ejecutado sin ambigüedades.

1.2.2 Un ejemplo cotidiano

Para comenzar a formalizar los conceptos fundamentales para escribir algoritmos veremos un ejemplo de la vida cotidiana:

Escribir la receta de cocina para hacer bombas de papa para 4 personas.

Existen dos fases en la resolución de este planteo:

- determinar los ingredientes necesarios,
- describir los pasos para hacer bombas de papas.

1. Ingredientes (elementos necesarios): 1 Kg. de papas, 1 huevo, pan rallado, queso cremoso, aceite, sal y pimienta a gusto.
2. Separar en tareas los pasos necesarios para lograr el objetivo:
    - Tarea 1: Pelar un Kg. de papas y cortarlas en dados pequeños
    - Tarea 2: Hervir las papas hasta que estén cocidas
    - Tarea 3: Pisar las papas hasta lograr un puré
    - Tarea 4: Condimentar a gusto y dejar enfriar.
    - Tarea 5: Batir el huevo en un bols
    - Tarea 5.1: Salpimentar a gusto
    - Tarea 6: Colocar en un plato el pan rallado
    - Tarea 7: Cortar el queso en dados de 1cm aproximadamente
    - Tarea 8: Tomar una porción de puré.
    - Tarea 8.1: Colocarle un dado de queso en el interior

- Tarea 8.2: Realizar una bola ocultando el queso
- Tarea 8.3: Pasarla por el huevo y luego por el pan rallado
- Repetir la tarea 8 hasta que no haya más puré.
- Tarea 9: Colocar una taza de aceite en una sartén
- Tarea 9.1: calentar a fuego moderado
- Tarea 10: Colocar las bombas de papas en el aceite y cocinar hasta que se doren
- Tarea 10.1: Sacar del fuego y colocarlas sobre papel absorbente.
- Repetir la tarea 10 para cocinar todas las bombas.
- FIN.

Seguramente Ud. habrá escrito una receta de cocina distinta, o habrá pensado que no sabe hacer este algoritmo, y por lo tanto debió consultar con algún cocinero **experto**. Experto no se refiere a un chef especializado, sólo debe ser una persona que sepa realizar las acciones, por ejemplo su mamá.

Pero ¿quién "ejecuta" estas acciones? Tal cual está planteado el algoritmo lo podrá ejecutar toda persona que comprenda que es "batir los huevos" sin ambigüedades ni más detalles. Tal vez Ud. pueda seguir estos pasos y terminar con una exquisita comida, o quizás no esté aún preparado para enfrentar dicho desafío.Pero porqué pensar en subestimar al lector, tal vez Ud. sea un experto cocinero, y no necesite seguir estos pasos para hacer las bombas de papas, simplemente su algoritmo sea *Tarea 1: hacer bombas de papas para 4* personas, pues es una orden que entiende perfectamente.Se concluye pues que la redacción de un algoritmo depende del procesador que lo va a ejecutar.

Como conclusión, frente a un problema debemos saber quién será el procesador y escribir una solución que pueda entender y llevar a cabo sin ambigüedades.

1.2.3 Problemas algorítmicos

Son aquellos problemas cuya solución puede expresarse mediante un algoritmo.

En la vida cotidiana encontramos muchos problemas algorítmicos, tanto dentro como fuera del campo altamente especializado de la informática. Otros problemas algorítmicos son los siguientes:

Problema algorítmico	**Algoritmo**
Ir a la Biblioteca.	Conjunto de acciones para llegar a la Biblioteca
Dar la vuelta al mundo por el Ecuador.	Un itinerario para recorrer el mundo
Resolver una ecuación matemática.	Pasos para resolver una ecuación
Cambiar la rueda a un auto.	Pasos a seguir para cambiar la rueda

En estos casos, los algoritmos muestran las principales características señaladas en la definición: exactitud, efectividad y terminación garantizada. Cuando se ejecutan determinados algoritmos pueden completarse en un pequeño intervalo de tiempo, mientras que otros pueden llevar mucho tiempo.Aquí cabe preguntar si existen problemas no algorítmicos, y la respuesta es afirmativa.

Un ejemplo de este caso es el problema de "Escribir todos los números enteros comenzando desde el 1". No es un problema algorítmico, pues la resolución (que todos podemos imaginar y escribir sin dificultad) no cumple la condición de finitud.

1.2.4 Definición de ambiente y acción

***Ambiente**de un trabajo, es el conjunto de todos los recursos necesarios para la ejecución de ese trabajo.*

Por ejemplo los ingredientes son el ambiente en la receta de cocina.

Acción *es un evento que modifica el* ambiente.

Descripto el ambiente de un problema, una acción sobre ese ambiente es un hecho de duración finita por el cual, a partir de un estado inicial, se obtiene un nuevo estado final. Por ejemplo, en la acción “pelar las papas”, el estado inicial es: las papas sin pelar y el estado final, las papas peladas.

1.2.5 Acción primitiva y no primitiva

A continuación veremos cuáles son las acciones primitivas y cuáles no lo son.

*Las **acciones primitivas** son las que el procesador puede interpretar y ejecutar.*

Pero, ¿qué sucedería si ese procesador no entiende una acción determinada? Específicamente esa acción no pertenece a su lenguaje. Habrá que modificarla de tal manera que pueda ser escrita en términos entendibles por el procesador. En muchos casos este tipo de acciones deben ser descompuestas en tareas más detalladas para lograr el efecto que se desea.

Por ejemplo en nuestro algoritmo culinario, podría ser el caso que un determinado procesador no sepa de que se trata la tarea: “Batir huevos en un bols” y haya que explicárselo con más detalles.

*La **acción no primitiva** debe ser descompuesta en acciones primitivas.*

Existen diferentes técnicas para descomponer acciones no primitivas en acciones primitivas. Uno de estos métodos se denomina ***Top-Down*** o de ***refinamientos sucesivos,*** y será el que usaremos en este libro.

Este método parte de una idea general de resolución y luego va descomponiendo las tareas hasta conseguir una secuencia de acciones primitivas que permitan al procesador resolverla.

Se concluye pues que se debe escribir un algoritmo que resuelva el problema dado, y que luego deberá ser adaptado al procesador que lo va a ejecutar. Por ello, frente a un problema debemos saber quien será el procesador y escribir una solución que pueda entender y ejecutar.

En este libro planteamos que el procesador será la ***computadora*** equipada con algún lenguaje específico.

La computadora puede trabajar con distintos lenguajes. Sin embargo, para un gran número de tareas de uso corriente, las acciones primitivas son similares en los distintos lenguajes, lo que permite desarrollar algoritmos que sean independientes del lenguaje e incluso de la máquina que los va a ejecutar.

1.2.6 Programa

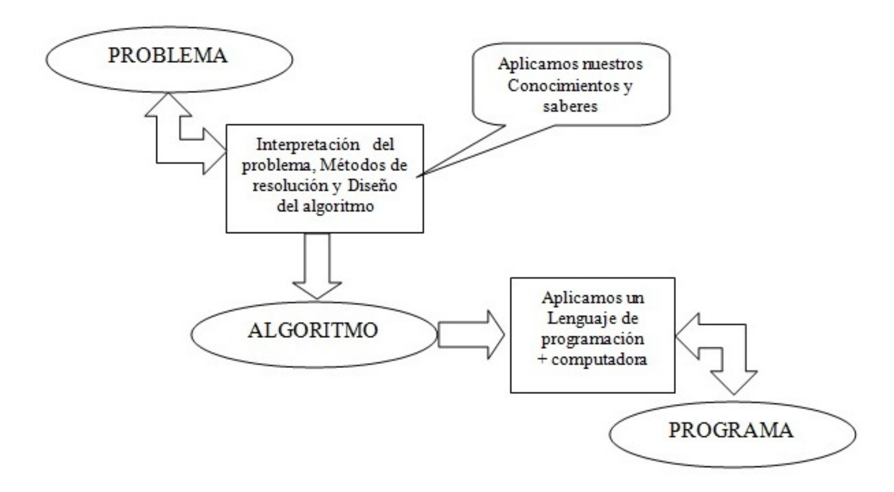
Definiremos qué se entiende por el término *programa*.

*Un **programa** es un conjunto de acciones que puede entender y ejecutar una computadora.*

Otra definición podría ser: “es un algoritmo traducido a algún lenguaje que pueda ser entendido por una computadora para poder ejecutarlo”. Cada acción del programa se denomina instrucción.

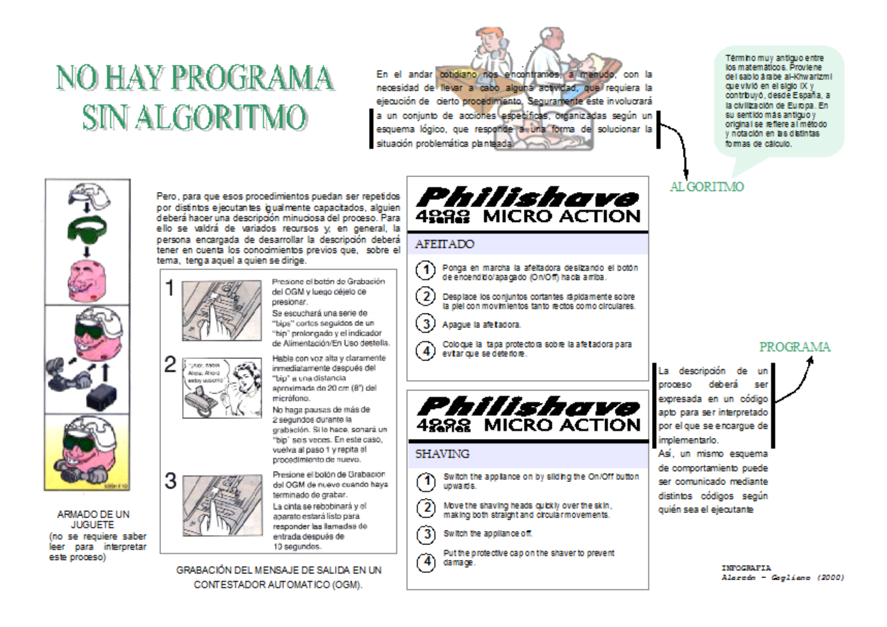
*Una **instrucción** es una combinación de palabras y símbolos que obedeciendo a la sintaxis propia de un lenguaje, son interpretados y utilizados por el computador para realizar una determinada acción.*

Para llegar a hacer un programa es necesario el diseño previo del algoritmo. Esto es, representar la solución del problema en un lenguaje natural, en el cual se expresan las ideas diariamente.



***Lenguaje de programación** es un conjunto de símbolos (sintaxis) y reglas (semántica) que permite expresar programas.*

Los algoritmos deberían ser independientes tanto del lenguaje de programación en que se expresa el programa, como de la computadora que lo va a ejecutar.



1.2.7 Representación de Algoritmos

Ahora veremos en que forma se puede escribir un algoritmo.
No existe un lenguaje único y universal para escribir algoritmos. Cada lenguaje tiene una sintaxis determinada, un grupo finito de elementos que ayudan a representar los algoritmos, y una semántica asociada a la sintaxis, la cual se refiere al concepto que se representa.

La sintaxis, conformada por dibujos y/o por palabras, no debe ser ambigua, y debe tener un nivel de detalle bien establecido con el cual pueda expresarse el algoritmo.
Otro punto importante a tener en cuenta es que toda representación de algoritmos debe usar alguna forma que permita independizar dicho algoritmo de detalles propios de los lenguajes de programación en sí. Además, la descripción del algoritmo deber ser tal que permita ser fácilmente transformado en un programa.
Existen dos tipos de representaciones: la gráfica (mediante dibujos) y la no gráfica (mediante textos).
Los métodos usuales para representar algoritmos son:

1. Diagrama de flujo
2. Diagrama de Nassi Schneiderman (comúnmente denominado Diagrama de Chapin)
3. Pseudocódigo
4. Lenguaje natural
5. Fórmulas

Los dos primeros son formas gráficas de representación, mientras que los últimos son no gráficas.

Analizaremos las distintas representaciones.

Los métodos 4) y 5) no suelen ser fáciles de transformar en programas. No es frecuente que un algoritmo sea expresado por medio de una simple fórmula, pues no es sencillo traducirlo a un lenguaje de programación. Observamos en el ejemplo de las bombas de papas lo ambiguo del lenguaje natural (en nuestro caso el español).
A continuación veremos las diferentes representaciones sobre un mismo algoritmo, cuyo problema base es:

“Calcular el promedio de un conjunto de números positivos, suponiendo que los números ingresan de a uno por teclado. Un valor cero como entrada indicará que finalizó la serie de números positivos.”

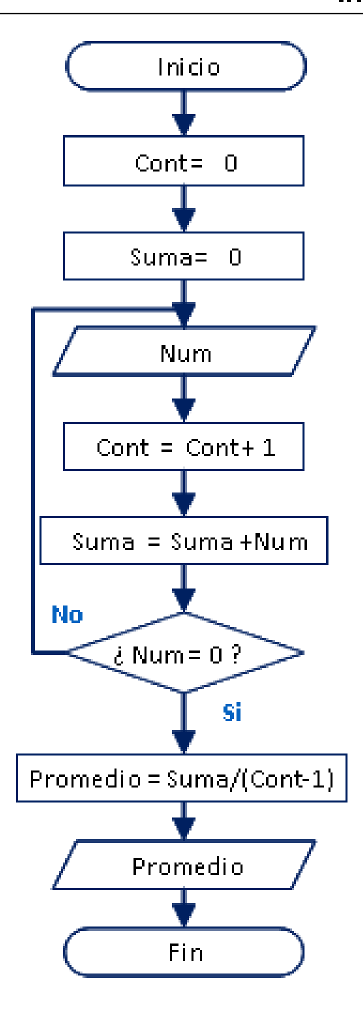
1.2.8 Diagrama de flujo

Es un diagrama que utiliza figuras geométricas unidas por flechas para indicar la secuencia en que deben ejecutarse las acciones. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI). Algunos de ellos y su significado son los siguientes:

Símbolo	Función
	Proceso, cualquier tipo de operación aritmética o de asignación
	Entrada o Salida de información desde o hacia los periféricos
	Decisión, indica operaciones de tipo lógico, preguntas, con respuesta SI o NO
	ĺnea conectora
	Indicador de dirección de flujo, indica el sentido de ejecución de las operaciones

El algoritmo del problema planteado utilizando un **diagrama de flujo** resulta de la siguiente manera.

Uno de los inconvenientes de este tipo de representación es la dificultad para modificarlo en el caso que se descubran errores de lógica y se debiera agregar dibujos para nuevas acciones.



Otra cuestión a notar es que no dispone de símbolos para las estructuras de control de repetición.

1.2.9 Diagrama de Nassi Schneiderman

También conocido como Diagrama de Chapin, es un diagrama de flujo donde se omiten las flechas de unión, resultando cajas contiguas. Las acciones se escriben en rectángulos sucesivos.

El algoritmo propuesto se representa en Diagrama de Chapin de la siguiente forma:

Cont← 0
Suma← 0
Leer (Num)
Mientras Num <> 0 hacer
Cont← Cont +1
Suma← Suma + Num
Leer (Num)
Promedio← Suma/Cont
Imprimir ("el promedio es:", Promedio)

Uno de los inconvenientes de este tipo de representación, al igual que el anterior, es la dificultad para modificarlo al agregar rectángulos para nuevas acciones si se descubren errores de lógica.

1.2.10 Pseudocódigo

Un pseudocódigo es un código no convencional o lenguaje no establecido universalmente.

Concretamente un pseudocódigo es un lenguaje formado por palabras comunes del idioma que elija el que lo use, en nuestro caso el español. Por lo cual el uso de tal tipo de lenguaje hace que la escritura de los algoritmos sea relativamente fácil.
La ventaja del pseudocódigo es que al usarlo, el programador se puede concentrar en la lógica de resolución del problema más que en las reglas del lenguaje, el cual ya conoce.
Por otra parte, facilita la modificación del algoritmo si se descubren errores de lógica, ya que para agregar nuevas acciones se escribe en el mismo renglón separando las acciones por un punto y coma.
El algoritmo propuesto se representa en pseudocódigo de la siguiente forma:

```
Algoritmo   Promedio_de_numeros_enteros
variables
        entero Num, Cont, Suma
        real Promedio
Inicio
        Cont = 0
        Suma = 0
        Leer (Num)
        Mientras  Num <> 0
                Cont = Cont + 1
                Suma =Suma + Num
                Leer (Num)
        fin mientras
        Promedio = Suma / Cont
        Escribir ('El promedio de los números es  ', Promedio )
Fin.
```

En este libro se utilizarán las representaciones Diagrama de flujo y Pseudocódigo indistintamente. Según nuestra experiencia es conveniente el Pseudocódigo a una representación gráfica, tal como Diagrama de Flujo o Diagrama de Chapín pues se considera que:

- Ocupa menos espacio en una hoja de papel.
- No se necesitan elementos de dibujo para realizarlo.
- Permite representar en forma fácil operaciones repetitivas complejas.
- Es muy fácil pasar del algoritmo representado por un pseudocódigo a un programa.
- Se puede observar claramente los niveles que tiene cada operación, por la indentación.
- Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.
- Es muy simple introducir modificaciones, continuando en el mismo renglón.

1.3 Lenguajes de programación

Como hemos visto, un Lenguaje de Programación es un conjunto de palabras y símbolos que pueden expresar ideas. Está compuesto por una sintaxis (palabras y símbolos) y una semántica (reglas que permite expresar ideas).

***Lenguaje de programación** es un conjunto de símbolos (sintaxis) y reglas (semántica) que permite expresar programas.*

Los lenguajes de programación se pueden clasificar de la siguiente manera:

- Lenguaje de máquina
- Lenguaje de bajo nivel
- Lenguaje de alto nivel

- Lenguaje algorítmico

A continuación veremos detalles de cada uno de ellos.

Lenguaje de máquina

Está escrito en un "idioma" que entiende el microprocesador (el CPU, el cerebro de la máquina). Las instrucciones son cadenas de 0 y 1 en código binario.Por ejemplo, la instrucción *sumar los números 9 y 11* podría ser 0001 1100 0111 0011 1001 1011

Ventaja: la velocidad de ejecución de los programas es directa pues el microprocesador las entiende y ejecuta.

Desventaja: es un lenguaje de símbolos muy diferentes al nuestro, por lo tanto la codificación del programa se hace lenta y se pueden cometer muchos errores. Las instrucciones en código de máquina dependen del hardware de la computadora y por lo tanto difieren de una a otra.

Lenguaje de bajo nivel

Los lenguajes de bajo nivel por excelencia son los ENSAMBLADORES o Lenguaje ASSEMBLER. Las instrucciones son mnemotécnicos, como por ejemplo ADD, DIV, STR, etc. Una instrucción del ejemplo anterior sería ADD 9,11.

Ventaja: mayor facilidad de escritura que el lenguaje anterior.

Desventaja: depende del microprocesador (existe un lenguaje para cada uno). La formación de los programadores es más compleja que la correspondiente a los programadores de alto nivel ya que exige no sólo técnicas de programación sino también el conocimiento del funcionamiento interno de la máquina. Se utiliza en la programación de PLC, control de procesos, aplicaciones de tiempo real, control de dispositivos. Un programa escrito en un lenguaje de este tipo no puede ser ejecutado directamente por el procesador, y debe ser traducido a código de máquina mediante un ensamblador.No se debe confundir, aunque en español adoptan el mismo nombre, el programa ensamblador *ASSEMBLER* encargado de efectuar la traducción del programa a código de máquina, con el lenguaje de programación ensamblador *ASSEMBLY LENGUAGE*.

Lenguaje de alto nivel

Está diseñado para que las personas escriban y entiendan los programas de modo mucho más natural, acercándose al lenguaje natural. Una instrucción del ejemplo anterior sería 9 + 11.Los lenguajes de alto nivel son los más populares y existe una variedad muy grande. Algunos ejemplos son BASIC, Pascal, C, C++, Cobol, Fortran, Delphi, Java, Python, SQL y todas sus variantes.

Ventajas: son los más utilizados por los programadores. Son independientes de la máquina, las instrucciones de estos lenguajes no dependen del microprocesador, por lo cual se pueden correr en diferentes computadoras. Son transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras. La escritura de los programas con este tipo de lenguajes se basa en reglas sintácticas similares a nuestro lenguaje, por lo tanto el aprestamiento de los programadores es mucho más rápida que en los lenguajes anteriormente nombrados.

Desventajas: Ocupan más lugar de memoria principal (RAM) que los anteriores. El tiempo

de ejecución es mayor pues necesitan varias traducciones.

Lenguaje algorítmico

Está formado por acciones primitivas provenientes de nuestro lenguaje natural. Si nosotros somos los que vamos a ejecutar esas acciones, entonces se podrá escribir en español. Una instrucción del ejemplo anterior sería 9+11=

Ventaja: es fácil aprender un pseudocódigo.

Desventaja: necesita muchas traducciones para ser ejecutado por el microprocesador, una de ellas es la conversión a un programa a través de un lenguaje de programación.

1.4 Nuestro procesador: la computadora

En el capitulo anterior hemos visto el modelo computacional (Entrada-Proceso-Salida) focalizando así el estudio en la computadora como procesador de los algoritmos que desarrollaremos a lo largo de este libro. A continuación analizaremos brevemente los conceptos básicos que hay que tener en cuenta para poder programar para ella.El diccionario de la Real Academia Española define computador electrónico como "Máquina electrónica, analógica o digital, dotada de una memoria de gran capacidad y de métodos de tratamiento de la información, capaz de resolver problemas matemáticos y lógicos mediante la utilización automática de programas informáticos." La propia definición nos da indicios acerca de algunos elementos básicos del computador:

- la memoria y
- algún dispositivo capaz de efectuar cálculos matemáticos y lógicos.

La **memoria** es un gran almacén de información. En ella se guardan todo tipo de datos: valores numéricos, textos, imágenes, sonido, etc.

El dispositivo encargado de efectuar operaciones matemáticas y lógicas, recibe el nombre de **Unidad Aritmético-Lógica** (UAL o ALU en inglés). Es una calculadora inteligente capaz de trabajar con datos y producir, a partir de ellos, nuevos datos, o sea el resultado de las operaciones.Existe otro dispositivo muy importante, la **Unidad de Control** (UC), que se encarga de transportar la información de la memoria a la UAL, de controlar la UAL para que efectúe las operaciones pertinentes y de depositar los resultados en la memoria.El conjunto que forman la Unidad de Control y la UAL se conoce por **Unidad Central de Proceso** (UCP o CPU en inglés Central Processing Unit).

Se puede imaginar la memoria como un armario enorme con cajones numerados y la UCP, como una persona que, equipada con una calculadora (la UAL), es capaz de buscar operandos en la memoria, efectuar cálculos con ellos y dejar los resultados en la memoria. Recordar que el procesador trabaja todo en la memoria.Utilizando un lenguaje más técnico: cada uno de los cajones que conforman la memoria recibe el nombre de*c***elda de memoria**y el número que lo identifica es su **dirección** *o* posición en la memoria.

Las celdas de la memoria son circuitos electrónicos organizados en pequeñas porciones, todas de igual tamaño. En cada una de ellas se puede almacenar secuencias de unos y ceros de tamaño fijo.

¿Por qué unos y ceros?

Porque la tecnología de las computadoras se basa en la sencillez de construir dispositivos binarios, con dos posibles estados: encendido/apagado, hay corriente/no hay corriente, cierto/-falso, uno/cero. ¿ Es posible representar datos tan variados como números, textos, imágenes, sonido, videos, con sólo unos y ceros? La respuesta es SI.

Cada uno de estos compartimentos o celdas de la memoria es un byte o unidad de información. Tiene asignada una dirección única, que le servirá a la Unidad de Control para organizar su tarea.

Por ejemplo:

Dirección	Contenido
....	
FFF8	5
FFF9	12
FFFA	L
FFFB	I
FFFC	B
FFFD	R
FFFE	O

Un byte es un conjunto de 8 bits. Un bit (acrónimo de las palabras **bi**nary digi**t**) es la representación de un estado de la señal eléctrica.

En este modelo de memoria, cada dirección indica unívocamente una celda. Sobre cada celda se puede leer o escribir. La operación de escritura sobre una celda, coloca un valor sobre la misma destruyendo el valor previo. Si en este caso se escribiese en la celda FFFC el valor 10, en una lectura posterior arrojaría el valor 10 y no la 'B' pues ésta ha sido reemplazada. Aquí podemos concluir que **la operación de escritura es destructiva del contenido de la celda, no así la lectura.**De este modo los dispositivos de entrada (teclado, mouse, archivo) almacenan (guardan, escriben) los datos en memoria, el procesador los procesa (realiza operaciones, lee y escribe en la memoria) y los dispositivos de salida (pantalla, impresora, archivo) muestran los contenidos de la memoria al mundo exterior.

La **MEMORIA PRINCIPAL** es el dispositivo de almacenamiento temporal de DATOS, INSTRUCCIONES y RESULTADOS intermedios y definitivos de la ejecución de programas. El término temporal se refiere a que este tipo de memoria tiene la característica de ser volátil, es decir, que cuando se corta el suministro eléctrico la información en ella desaparece.

Por otro lado se encuentran las memorias permanentes, denominadas **MEMORIAS SECUNDARIAS**, las que permiten guardar de forma permanente programas e información. En este rubro se encuentran los discos y las memorias flash, entre otras.

1.5 Conceptos y elementos básicos para la resolución algorítmica

Hemos visto que el manejo de la memoria de una computadora es de suma importancia. En esta línea, una cuestión a tener en cuenta son los tipos de dato que maneja el computador:

números y caracteres.Los números pueden estar escritos en sistema decimal (nuestro sistema diario), sistema binario (utilizado por el microprocesador por conveniencia y simplicidad), o sistema hexadecimal (notación compacta del binario). Por otro lado están los caracteres, sus símbolos, representaciones y significados. Un carácter es una letra mayúscula, minúscula, signo de puntuación, signo especial, etc.A continuación veremos a que se refiere un tipo de dato y presentaremos las variables y las constantes, conceptos basales de los algoritmos.

1.5.1 Tipo de dato

Definiremos tipo de dato de la siguiente manera:

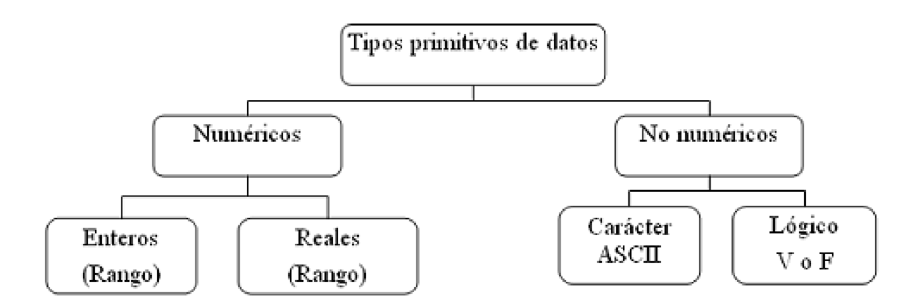
Un ***tipo de dato*** *está determinado por un conjunto de valores ordenados y por las operaciones permitidas sobre esos valores.*

Los **atributos** que distinguen a un tipo de otro son:

- Conjunto de valores ordenados y rango de definición de los mismos
- Conjunto de operaciones que se pueden realizar sobre ellos
- Representación interna de máquina de los valores.

Se debe notar que, al contrario de lo que ocurre en el Álgebra, el conjunto de valores es **finito**, tiene un **rango de valores**, o sea un mínimo y un máximo claramente especificados.

Los tipos de dato que trabajaremos para nuestro procesador son ***entero****,* ***real****,* ***carácter*** *y* ***lógico.*** *S*e los conoce como **tipos primitivos de datos.**



El conjunto de valores que pueden asumir los enteros o los reales están acotados a los rangos de los números que puede manejar el procesador en cuestión, con las limitaciones de bytes que se usen, es decir, del contenedor que disponga ese procesador.Por otro lado están los caracteres, sus símbolos, representaciones y significados. Cada carácter (letras mayúsculas, minúsculas, signos de puntuación, signos especiales, etc.) tiene asociado un código numérico. Estos valores numéricos fueron establecidos en una codificación estándar, el Código ASCII (American *Standard Code for Information Exchange*). Esta codificación utiliza 7 bits para representar un carácter, a cada uno de los cuales le corresponde un número entre el 0 y el 255. A continuación se observa la tabla ASCII, donde cada columna consta de un número de codificación del carácter correspondiente:

1.5.2 Tipos de dato a utilizar

Los tipos de dato que reconoce nuestro procesador, la computadora, son los “enteros”, los “reales”, los “caracteres” y los “lógicos”. La tabla siguiente muestra ejemplos de rango de valores y operaciones permitidas de cada tipo primitivo.

Otro tipo de dato que se utiliza habitualmente es el **string**, el cual se refiere a una cadena de caracteres, es decir texto. Por ejemplo, un nombre o una dirección. Un objeto del tipo de dato

0		32		64	@	96	`	128	Ç	160	á	192	└	224	α
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Γ
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	π
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	Σ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	σ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	╞	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	╟	231	τ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Φ
9	○	41	)	73	I	105	i	137	ë	169	⌐	201	╔	233	Θ
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Ω
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	δ
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	∞
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	φ
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	ε
15	☼	47	/	79	O	111	o	143	Å	175	»	207	╧	239	∩
16	►	48	0	80	P	112	p	144	É	176	░	208	╨	240	≡
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	╤	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	╥	242	≥
19	‼	51	3	83	S	115	s	147	ô	179	│	211	╙	243	≤
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	╘	244	⌠
21	§	53	5	85	U	117	u	149	ò	181	╡	213	╒	245	⌡
22	▬	54	6	86	V	118	v	150	û	182	╢	214	╓	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	╖	215	╫	247	≈
24	↑	56	8	88	X	120	x	152	ÿ	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	∙
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	¢	187	╗	219	█	251	√
28	∟	60	<	92	\	124	¦	156	£	188	╝	220	▄	252	ⁿ
29	↔	61	=	93	]	125	}	157	¥	189	╜	221	▌	253	²
30	▲	62	>	94	^	126	~	158	₧	190	╛	222	▐	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

Tipo de dato	Valores	Rango	Operaciones
Entero	-3,-2,-1,0, 1, 2,3	Depende del microprocesador	+ - * / comparación
Real	-3.0, …… 3.0	Depende del microprocesador	+ - * / > < comparación
Carácter	Letras minúsculas, mayúsculas, símbolos, etc.	código ASCII	Comparación y concatenación
Lógico	V o F	V y F	Producto suma negación lógica

string asume como valores posibles secuencias de caracteres tomados de un conjunto específico (por ejemplo el código ASCII) y admite como operaciones la comparación y la concatenación. Cabe aclarar que el string no es un tipo de dato primitivo, sino un tipo de dato compuesto por elementos primitivos, los caracteres.Otro tipo de dato es el de los números **complejos**, formado por un par ordenado de números reales. Este tipo de dato compuesto es útil en aplicaciones de Ingeniería y permite realizar directamente las operaciones sobre números complejos. Conjuntos

de datos más elaborados, con características y operaciones propias constituyen los tipos de dato compuestos o **estructuras de datos**, las cuales veremos en un capítulo posterior. Los strings y números complejos son casos de estructuras de datos.

1.5.3 Variables y constantes

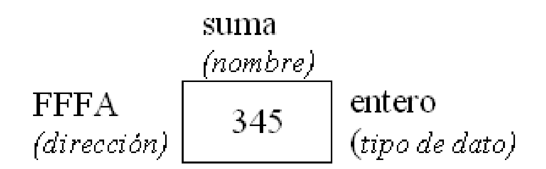
A continuación veremos dos conceptos importantes de la Algoritmia: variables y constantes.El procesador necesita disponer de un mecanismo que permita el almacenamiento y la manipulación de los datos. En un algoritmo o programa esto es llevado a cabo por entidades a las que denominaremos ***variables*** y ***constantes***. Definimos:

*Una **variable** es un objeto de memoria cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa*

Los **atributos** de una variable son **nombre, tipo de dato y valor.**

- Un **nombre** que la designa (en bajo nivel hace referencia a la dirección de una celda de memoria). Es recomendable que el nombre de la variable sea representativo del uso de la misma a lo largo del algoritmo o programa.
- Un**tipo de dato** que describe el uso de la variable.
- Un **valor** que describe el contenido.

El siguiente ejemplo da cuenta del nombre de la variable "suma", del tipo de dato que puede manejar "entero" y de la ubicación en memoria "FFFA".



Existen lenguajes donde es obligatorio **declarar las variables a utilizar**, es decir, listar las variables que se utilizarán e indicar el tipo de cada una de ellas. Estos lenguajes son de tipo fuerte o estricto como por ejemplo el lenguaje Pascal. Mientras que en otros lenguajes como C, Python, Fox, no hace falta una declaración del tipo de dato de las variables, aunque siempre es recomendable por legibilidad del algoritmo o programa.

La **declaración** *de las variables implica darles un lugar en memoria, un nombre y tipo de dato asociado.*

En nuestro caso realizaremos la declaración de las variables previamente al algoritmo, considerando a éstas como el ambiente de resolución del problema. Cuando se declara una variable, el microprocesador reserva una porción de memoria principal para almacenar los valores correspondientes al tipo de dato de la misma.

Para unificar los criterios de escritura de los algoritmos escribiremos primero el tipo de dato y luego la lista de variables separadas por coma

Por ejemplo:

real	suma, perímetro
caracter	inicial
entero	edad, comisión, cantidad, contador

Dado el concepto de variables ahora veremos qué es una constante.

Una constante es un objeto cuyo valor no *puede ser modificado durante el desarrollo del algoritmo o ejecución del programa.*

El concepto “constante” es similar al de variable pero con la particularidad de que su valor permanece inalterable.

La **definición** *de las constantes implica darle un nombre a un valor determinado.*

Con una constante se especifican valores que no se calculan ni se leen, sino que permanecen constantes. En el ejemplo del cálculo del perímetro de la circunferencia (2*Pi*radio), “2” es una constante expresada por su valor, “radio” es una variable y “Pi” es una constante expresada con un nombre, la cual fue definida como Pi=3,1416 al comienzo del algoritmo. Cada vez que el nombre Pi aparece en el desarrollo del mismo, será reemplazado por el valor 3,1416.

1.5.4 Nombres de variables/constantes (identificadores)

Cada lenguaje tiene reglas para construir los nombres o identificadores. En este libro vamos a proponer un conjunto de reglas propias, a saber:

1. Todo identificador debe comenzar con una letra del alfabeto inglés (mayúscula o minúscula),
2. No debe ser palabra reservada del lenguaje,
3. No debe contener espacios en blanco ni punto,
4. Puede contener una combinación de los siguientes caracteres:
   - Letras: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   - Dígitos: 0 1 2 3 4 5 6 7 8 9
   - Caracteres especiales: _ @ / ( ) ’ ! ” % & < > ? $
5. La cantidad de caracteres que conforman al identificador no está limitada,
6. Todo identificador debe representar el uso que se le dará en el algoritmo.

Los siguientes son ejemplos de nombres de identificadores válidos:
Curso_de_Informatica r2e2 Info INFO iNfO

Los siguientes son ejemplos de nombres de identificadores no válidos:
@comienzo comienza con un símbolo especial
La solución contiene un espacio en blanco y una vocal acentuada
1er_solucion comienza con un número

1.5.5 Clasificación de Variables

Las variables se pueden clasificar por su contenido y por su uso.

Por su Contenido	**Por su Uso**
Numéricas	de Trabajo
Lógicas	contador
Carácter	acumulador o sumador
String	

Por su Contenido

Variables Numéricas: son aquellas en las cuales se almacenan valores numéricos, enteros o reales, positivos o negativos. Ejemplo: costo=2500 , x=12,50Una variable numérica almacena números y sólo puede estar involucrada en operaciones aritméticas.*Variables Lógicas:* son aquellas que sólo pueden contener uno de 2 valores: *verdadero* o *falso (V o F, true o false, 0 o 1, Si o No).* Ejemplo z= V , x= F*Variables Car*ácter*:* cuyo contenido es un solo carácter alfanumérico (una letra, un carácter especial, un número), que escribiremos entre apóstrofo. Ejemplo: letra = ’a’, Inicial = ’L’, Rta = ’ S ’, dígito= ’4’*Variables Cadena de caracteres o String:* cuyo contenido está

formado por caracteres alfanuméricos (letras, números y caracteres especiales), que escribiremos entre comillas. Ejemplo: letra="a", Apellido="Lopez", Direccion="Pasco #190"

En este punto es interesante analizar la siguiente pregunta: ¿Cuál es la diferencia que existe entre las constantes "a" y 'a'? En primer lugar, la "a" es una constante de tipo string, pues est'a escrita entre comillas, y la 'a' es de tipo car', pues est'a entre ap'ostrofos. Pero hasta aquí no notamos grandes diferencias. Entonces la pregunta es emph¿Qué diferencia hace el procesador con los caracteres y los strings? El procesador utiliza un Byte para almacenar un carácter. Sin embargo, cuando se trata de string, dependerá de la longitud del mismo. Por ello, el microprocesador agrega un carácter especial para marcar el final de la cadena de caracteres. Cuando el procesador almacena una cadena de caracteres en memoria lo hace ubicando cada carácter en un Byte en posiciones consecutivas de memoria. Pero cuando quiera recuperar dicha cadena de memoria no sabrá hasta que Byte debe tomar. Este problema se solucionó haciendo que el procesador coloque un carácter especial al final de la cadena para identificar su fin. A este car'acter especial lo denominaremos emphfin de cadena, y como es especial no figura en el código ASCII. Para poder expresarlo en nuestros algoritmos lo escribiremos como '/s' , aunque usaremos 2 caracteres se considerará que internamente el procesador lo va a entender como el carácter fin de cadena. Este carácter ocupa un Byte.En conclusión, la diferencia entre laemph "a" emph y laemph 'a', es que la "a" ocupa dos Bytes de memoria, uno para la representación del cáracter a y otro para el fin de cadena, en cambio la 'a' ocupa un único lugar de memoria. En el caso de variables de tipo cadena el nombre de la variable indica la posición inicial y el resto es consecutivo.

Por su Uso

Variables de Trabajo: son variables que reciben un valor, ya sea de un ingreso de información o como resultado de una operación. Ejemplo: suma = a + b, precio= 10.52*Contadores:* son variables que se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Los incrementos son generalmente de uno en uno. Ejemplo: Cont = Cont + 1*Acumuladores o Sumadores:* son variables que se utilizan para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente. Ejemplo: Suma = Suma + x

1.5.6 Expresiones

En este ítem veremos en qué operaciones pueden intervenir los datos de acuerdo a su tipo.

*Una **expresión** describe un cálculo a efectuar cuyo resultado es un único valor.*

Una expresión está constituida por operandos y operadores.

Las expresiones pueden ser del tipo numérica, relacional, lógica o de carácter, de acuerdo al tipo de dato de su resultado. El resultado de una expresión aritmética es del tipo numérico, el de una expresión relacional y el de una expresión lógica es de tipo lógico, y el de una expresión carácter es del tipo cadena de caracteres o carácter.

Expresiones aritméticas

Las expresiones aritméticas son análogas a las fórmulas matemáticas. Los elementos intervinientes son numéricos (reales, enteros) y las operaciones son aritméticas.

Los operadores son:

(*) Cuando en una operación los dos operandos son enteros el resultado es entero. Pero si al menos uno de los operandos es real, el entero interviniente se flota, es decir, se convierte en notación de punto flotante, y así el procesador opera en real, dando como resultado un número real.

Operador	Significado	Tipos de operandos	Tipo de resultado
+	Suma	Enteros o reales	Entero / real (*)
-	Resta	Enteros o reales	Entero / real (*)
*	Multiplicación	Enteros o reales	Entero / real (*)
/	División real	Enteros o reales	real
**	Potencia	Enteros	Entero / real (**)

En forma similar ocurre si algún operando es complejo, en cuyo caso el resultado es complejo.

(**) En la operación de potencia se consideran ambos operandos como enteros. Cuando el resultado se sale del rango de enteros se lo registra como un número real.
Reglas para evaluar expresiones aritméticas Las expresiones se evalúan de izquierda a derecha. Regla algorítmica fundamental: la expresión debe estar escrita en un único renglón.Los paréntesis se usan para anidar expresiones y alterar el orden de evaluación. Las operaciones encerradas entre paréntesis se evalúan primero.Las *reglas de precedencia* de las expresiones indican el orden en que la computadora evalúa una expresión cuando hay más de un operador. Para expresiones aritméticas se siguen, lógicamente, las reglas de precedencia aritmética. El orden de evaluación de los operadores en cualquier expresión es el siguiente:

- Paréntesis (empezando por los más internos)
- Potencias
- Productos y Divisiones
- Sumas y restas
- Concatenación
- Relacionales
- Lógicos

Cuando hay dos operadores con la misma precedencia, se calcula primero la operación que está a la izquierda.

Funciones internas o de biblioteca
Los lenguajes de programación traen incorporado funciones que conforman las funciones de biblioteca (library en inglés), algunas de ellas se muestran en la siguiente tabla:

Ejercicios propuestos
¿Cuál es el resultado de las siguientes expresiones algorítmicas aritméticas?

- a) 10.5 / 3 =
- b) 1/ 4 =
- c) 1/ 4.0 =
- d) 3 + 6 * 2 =
- e) -4 * 7 + 2 ** 3 / 4 - 5 =
- f) 12 + 3 * 7 + 5 * 4 =
- g) sqrt(25) =
- h) abs (6) =
- i) abs (-12.5) =
- j) nint(6.64) =
- k) nint (6.23) =
- l) int(6.64) =

Función	Descripción	Ejemplo
abs (x)	Valor absoluto	abs (x)
sqrt (x)	Raíz cuadrada	sqrt (9)
cos (x)	Coseno de x, x en radiantes	cos (alfa)
sin (x)	Seno de x, x en radianes	sin (alfa)
exp (x)	Exponencial	exp (5)
int(x)	Parte entera de x	int(5.70)
Nint(x)	Redondea al entero más próximo	Nint(23.52)
Mod(div, divis)	Resto de la división entera	mod(10,3)
asin(x)	Arco seno de x, x en radianes	asin(x)

- m) int(6.23) =
- n) (2 * X) – 5 /= (3 + 8) * 2 = si X=3

Convertir las siguientes expresiones algebraicas en expresiones algorítmicas. Tener en cuenta la regla fundamental de la escritura de las expresiones en Algoritmia: ***deben estar escrita en un único renglón.***

a) $5(x+y)$ →

b) $\dfrac{x+y}{u+\dfrac{w}{a}}$ →

c) $\dfrac{-b \pm \sqrt{b^2-4ac}}{2a}$ →

Expresiones relacionales y lógicas

El resultado de este tipo de expresiones es de tipo lógico: Verdadero o Falso.

Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas mediante operadores lógicos (NOT, AND, OR en inglés; NO, Y, O en español) y/o operadores de relación (< > <= >= = < >).

Una expresión del tipo relacional tiene operandos que admiten una relación de orden, tales como los números enteros, caracteres y cadenas de caracteres. Una expresión del tipo lógico, también denominado *booleano*, tiene como operandos expresiones relacionales y/o expresiones lógicas. Los operadores lógicos son: el NOT que es unario, es decir, aplicado a un único operando, el AND y el OR, los cuales necesitan 2 operandos.

Operadores de relación > mayor < menor = igual >= mayor igual <= menor igual < > ó /= distinto

Ejemplo Si A=4 y B=3, entonces (A-2) < (B-4) resulta falso.

Para realizar comparaciones de datos de tipo carácter, se requiere un orden predefinido de

los caracteres. Para ello recurrimos al código ASCII. Allí podemos ver que la ‘a’ es mayor que la ‘ A ‘, es decir, está antes la mayúscula que la minúscula.

Ejercicio propuesto

Determinar el resultado de las siguientes expresiones relacionales, considerando la lógica del código ASCII:

‘A’ > ‘a’ = Tomás’ > ‘Tomar’ = ‘libro’ < ‘librería’ =

Operadores lógicos

1- **NOT** es un operador unario, es decir que influye sobre una única expresión del tipo lógica.

NOT (expresión lógica)

NOT (verdadero) = Falso
NOT (falso) = Verdadero

2- **AND** es la conjunción, la multiplicación lógica, cuya lógica es:

expresión1	expresión2	exp1 .AND. exp2
V	V	V
F	F	F
V	F	F
F	V	F

Conclusión: el resultado de una expresión con un operador AND es *verdadero* siempre y cuando todas las expresiones son verdaderas, en caso de existir una expresión falsa, el resultado es falso.3- **OR** es la disyunción o suma lógica, cuya lógica es:

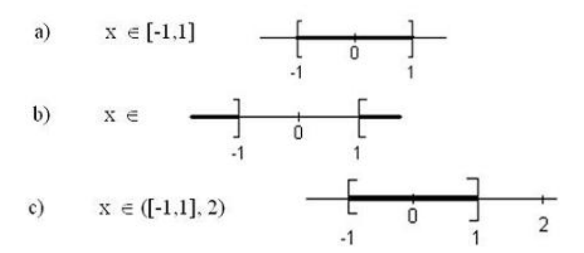
expresión1	expresión2	exp1 .OR. exp2
V	V	V
F	F	F
V	F	V
F	V	V

Conclusión: el resultado de una expresión con un operador OR es *falso* siempre y cuando todas las expresiones son falsas, en caso contrario es verdadero.

Ejercicios propuestos

¿Cuál es el resultado de las siguientes expresiones algorítmicas lógicas?
.NOT. 4 > 6 = (1.0 < X) AND (X < Z + 7.5) = considerar que X= 7 y Z = 4 5 < B – 4 .AND. A = 4 = si A=4, B=3 X – 2 = 1 .OR. (X+3) > 6 = si X=3 .NOT. Z > 6 = si Z=6

¿Cómo se expresan las siguientes condiciones algebraicas en forma algorítmica?



Expresiones de caracteres

Una expresión del tipo cadena de caracteres involucra operandos del tipo cadena y admite la operación de concatenación. Se identifica por un signo + y los operandos se encierran entre comillas dobles.
Por ejemplo la operación " Hola " + "amigo" da como resultado " Hola amigo".

1.6 Algoritmos en Pseudocódigo

En la sección anterior describimos los tipos de dato que maneja nuestro procesador, el concepto de variable y de constante, y cómo se conforman las expresiones.

En esta sección comenzaremos a conocer la sintaxis y semántica del lenguaje que entiende nuestro procesador. Veremos algunas de las acciones primitivas del lenguaje.

Generalizando, las acciones que realiza nuestro procesador son:

- "toma" información desde algún periférico de entrada,
- actualiza la memoria de datos,
- procesa los datos para obtener resultados parciales y finales (suma, resta, compara)
- "envía" los resultados a algún periférico de salida.

1.6.1 Acción de asignación

El objetivo de una acción de asignación es cambiar el valor almacenado en una variable.

Sintaxis <**variable**> ← <**expresion**>

La expresión es evaluada, y su resultado es asignado como valor de la variable a la cual apunta la flecha, es decir, la expresión de la derecha se evalúa y su resultado es copiado al nombre de variable de la izquierda.Ejemplos:

Sean entero i ; carácter c

i ←3 + 4 se le asigna el valor 7 a la variable i

c ←' T ' se le asigna el valor T a la variable c

Así, tendremos asignación aritmética, lógica, de carácter y de cadena de caracteres.

Notar que:

- La ocurrencia de una variable en el lado izquierdo de una asignación denota la *posición de memoria* donde guardar el valor resultante de evaluar la expresión del lado derecho.

- La ocurrencia de una variable en el lado derecho de una asignación denota su *valor* actual.
- Una misma variable puede aparecer en la parte izquierda y derecha de una asignación.

Por ejemplo: **x** ←**x + 1**

Esta expresión es muy usual en la resolución de algoritmos, pero **NO** debe interpretarse como una ecuación matemática, ya que no tendría sentido. Por este motivo se usa una flecha en vez de un signo de igualdad. Esta asignación significa que estamos usando el valor *actual* de la variable **x** para calcular su *nuevo* valor. **Corrección de tipo durante la asignación**
En una asignación **x**← **exp** , el *tipo* de la variable **x** y el de la expresión **exp** debe ser el mismo.

Pero existe una excepción, es posible asignar un valor *entero* a una variable *real*.

Ejemplo Si **x : real y z : entero**

z ← **4**

x ← **z**

Al asignar un valor entero a una variable real, el valor se transforma a un real, y luego se lo asigna a x, resultando **x = 4.0**La conversión de entero a real se realiza en forma automática, y se dice que el entero "se flota" (haciendo referencia al punto flotante, la coma decimal)La conversión de real a entero, es decir z ← x no es permitida y da error.

Ej. ¿Cual es la diferencia?
entre A ← B y A ← 'B'
entre A ← 7 y A ← '7'

1.6.2 Acción Leer

El objetivo de una *acción Leer* es cambiar el valor almacenado en una variable a través de periféricos de entrada (supongamos teclado, modem o disco). Se puede pensar como una asignación externa. Sintaxis **Leer (lista de variables)**

La lista de variables es separada por comas.

Ejemplos Leer (Num1, Num2)

Si desde teclado se ingresan dos número, el primer número ingresará en la variable Num1 y el segundo en la variable Num2.Toda asignación es una acción destructiva del contenido de memoria.

1.6.3 Acción Escribir

El objetivo de una *acción Escribir* es mostrar los resultados o textos a través de periféricos de salida (supongamos monitor, impresora, modem o disco)Sintaxis **Escribir (lista de expresiones de salida)**

La forma para usarla será:

Escribir (Num1) muestra el valor de una variable , en este caso de Num1

Escribir (“Buen día”) muestra el texto encerrado entre comillas

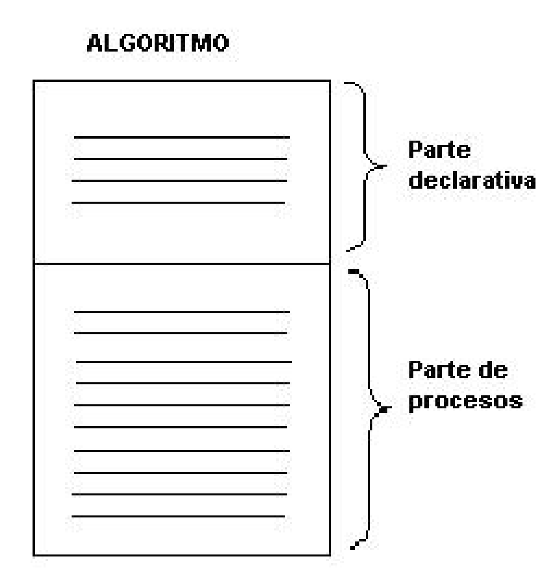
Escribir (“El resultado es “, Num2) muestra el texto y el valor de la variable

Escribir (Num1 * Num2 / 34) muestra el resultado de la expresión

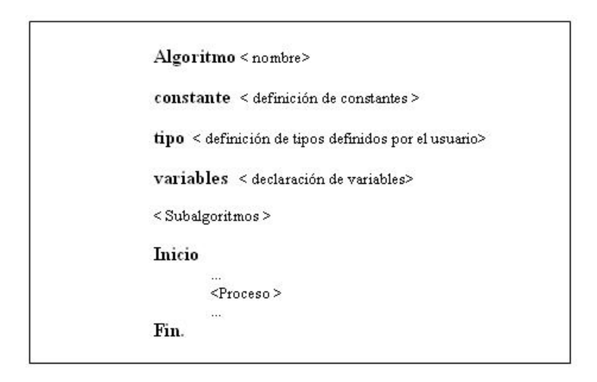
A continuación formalizaremos la escritura de un algoritmo.

Estructura de un algoritmo

Los algoritmos tienen dos partes bien definidas: la ***Parte declarativa,*** donde se describen los datos que se van a utilizar en el procedimiento de resolución (el ambiente) y la ***Parte de procesos***, donde se describen las acciones del algoritmo en sí (la lógica de resolución).



Existen variadas formas de escribir algoritmos en Pseudocódigo, en este libro nos guiaremos con la siguiente estructura:



Algoritmo < nombre>

constante < definición de constantes >

tipo < definición de tipos definidos por el usuario>

variables < declaración de variables>

< Subalgoritmos >

Inicio

...
<Proceso >
...

Fin.

Las palabras que están en negrita son palabras reservadas del lenguaje.
Las secciones **Algoritmo** < nombre>, **Inicio,** < Proceso > y **Fin.** (con punto al final) deben estar presentes en todo algoritmo, no así las secciones **constante**, **tipo, variables** y <Subalgoritmos> que dependerán de la construcción de la resolución. El tema Subalgoritmos se verá más adelante.

Parte declarativa

Ahora estamos en condiciones de concretar la forma en la cual se va a expresar la *Parte declarativa* de un algoritmo. Se deben listar todas las constantes con sus valores asociados, luego la lista de variables que se usarán en el proceso del algoritmo, especificando el nombre de la variable y el tipo de datos asociado. Y, si fuese necesario, la definición de tipos de dato creados por el programador.Ejemplos:

constante
Pi = 3,1416
IVA = 21
tipo
Dia_de_la_semana = (“lunes”,”martes”,”miercoles”,....”domingo”)
variables
real Suma, N1, N2
entero Edad
string Nombre, Dirección, Ciudad
Dia_de_la_semana dia

Recomendamos la lectura de *El estilo de la programación* pág 246 a 258 del libro “Fundamentos de Algoritmos y Programación”, autores Lage-Cataldi-Salgueiro.

Resolución del problema

La resolución de problemas tiene pasos específicos que seguiremos en el siguiente ejemplo:

“Realizar el algoritmo para determinar la superficie y el perímetro de terrenos rectangulares”

Como comentamos en la sección *Del mundo real a la solucion por computadora*, se deben seguir los siguientes pasos para resolver un problema dado:

- Interpretar el enunciado
- Describir Datos y Resultados
- Buscar una metodología de resolución
- Escribir el algoritmo de acuerdo a los puntos anteriores.

A continuación realizaremos el Análisis del problema planteado.

Análisis del problema

- **Datos**: frente y profundidad
- **Resultados**: Perímetro y superficie.
- **Metodologia de resolución**:
- Perímetro = 2 x (frente + profundidad)
- Superficie = frente x profundidad

Luego escribimos el algoritmo respetando lo diseñado en el Análisis del problema.

Algoritmo

En el próximo capitulo seguiremos viendo otros elementos que intervienen en los algoritmos.

```
Algoritmo  Cálculo de la superficie y el perímetro de terrenos rectangulares

variables
real  frente, profundidad, sup, perim

Inicio
          Escribir ("Ingrese la medida del frente")
          Leer ( frente)
          Escribir ("Ingrese la medida de la profundidad" )
          Leer( profundidad)
          Sup= frente * profundidad
          Perim = (frente + profundidad) * 2
          Escribir ( "El perímetro del terreno es ", perim , "y su superficie es ", sup)
Fin.
```

2 — Técnicas de programación. Organización de las acciones

Cristina I Alarcón

El problema que se plantea puede ser modesto; pero si pone a prueba la curiosidad que induce a poneren juego las facultades inventivas, si se resuelve por propios medios, se puede experimentar el encantodel descubrimiento y el goce del triunfo

George Polya

2.0.4 Introducción

Muchas veces, encontrar la solución a un problema no resulta una tarea sencilla. Primeramente hay que comprender bien cuál es el problema que se nos plantea, al cual le debemos encontrar su solución. Luego tendremos que identificar bien a la información que nos explicita el enunciado del problema, es decir identificar los datos. Una vezfijados los datos debemos deducir del enunciado cual es el o los resultados que nos debe brindar la solución del problema.
Cumplimentadas estas dos etapas: identificación de datos e identificación de resultado, recién estaremos en condiciones de comenzar a desarrollar una estrategia solución. Cuando la estrategia solución sea una secuencia de acciones, exactas, precisas y finitas que nuestro procesador (ente: persona o máquina) puede ejecutar con el solo hecho de enunciarlas, estaremos formulando el algoritmo solución de nuestro problema.

2.1 Técnicas de programación algorítmica

La elaboración de un algoritmo se puede realizar en forma totalmente libre, sin seguir los lineamentos de ningún modelo o en caso contrario aplicando las orientaciones de un prototipo.En el desarrollo de un algoritmo siguiendo un modelo establecido podemos citar, entre otros, a los siguientes modelos:

2.1.1 Modelo declarativo

En este modelo se declara una serie de proposiciones, en general todo tipo de transformaciones que relatan el problema e individualizan su solución. Se indica qué es lo que se quiere obtener, no los pasos necesarios para obtener la solución.Se trabaja por medio de cláusulas que responden a la particularidad *que afirmando se afirma*, llamado **modus ponens (latín)** .El **modo ponens** se basa en una regla de deducción que podría representarse de la siguiente manera:
Si proposicion1, *entonces* proposicion2
Proposicion1
Por lo tanto, proposicion2

Por ejemplo, un razonamiento que sigue las inferencias del modo " afirmando se afirma":

Si está nublado, entonces no llega el sol.
Está nublado.
Por lo tanto, no llega el sol.

En los algoritmos escritos en este modelo sus acciones no se ejecutan secuencialmente, no existen formas estructurales. Los algoritmos se fundamentan en dos nociones: la unificación y la vuelta atrás. La unificación establece que cada proposición determine un conjunto de nuevas proposiciones susceptibles de ser ejecutadas. La vuelta atrás permite regresar al punto de partida.Algunos lenguajes de programación que responden a este paradigma son, por ejemplo: Prolog, Lisp, Maude, Sql.

2.1.2 Modelo imperativo

En este modelo se detallan todos los pasos necesarios para encontrar la solución del problema. Las acciones se ejecutan secuencialmente, siguiendo una estructuración. Esta estructuración se implementa a través de las estructuras de control. Ejemplo:

Inicio
Acción 1
Acción 2
Acción 3

Hacer n veces

Si expresión es verdadera entonces
  Acción 4
  Acción 5
  Acción 6
Sino
  Acción 7
  Acción 8

Fin selección

Fin repetición

Acción 9
Acción 10
Fin

Entre los lenguajes que responden a esta forma de desarrollar programas podemos mencionar, entre otros: Fortran, Pascal, C, etc.

Nosotros utilizaremos este modelo aplicando una técnica específica que se desarrollará más adelante.

2.1.3 Modelo orientado a objetos

Los algoritmos que siguen este modelo se caracterizan porque tienen en cuenta las relaciones que existen entre todos los objetos que intervienen. Cada objeto o entidad que interviene en la solución tiene una determinado conducta, estado e identificación.En este modelo no se debe preguntar: **¿qué hace el algoritmo?, sino preguntar: ¿quién o qué lo hace?.**
Algunos lenguajes de programación que siguen este modelo son: Smalltalk, C++, HTML, Java, etc.

2.2 Estructuras de Control. Programación Estructurada

En la construcción de un algoritmo las acciones que lo integran deben agruparse de la forma que dicha resolución lo exija.Existirán para un mismo algoritmo muchas formas de organizar sus acciones, primitivas o no, que conduzcan a la solución requerida.Pensaremos en esas organizaciones como no excluyentes y que se podrán incluir o asociar entre ellas.Utilizaremos solamente tres únicas formas de disposición de las acciones.Hay veces que las acciones necesitan agruparse o estructurarse en forma de serie o sucesión. Otras veces las acciones deberán reunirse en un proceso repetitivo. En distintas ocasiones la agrupación de las acciones deberáresponder a un proceso de selección, que divide el camino para agrupar las acciones siguientes entre dos o más opciones.

2.2.1 Estructuras de control

Llamaremos **estructuras de control** a las organizaciones que controlan la ejecución de las acciones en un algoritmo.
Las **estructuras de control** son las que establecen el orden de ejecución de las acciones.
Permiten especificar la coordinación yregulacióndelalgoritmo, porque dirigen la dirección que debe seguir el flujo de información en el mismo.

2.2.2 Programación estructurada (PE)

La programación estructurada es una forma de desarrollar programas (algoritmos) que responde al modelo imperativo. Es un conjunto de técnicas para desarrollar algoritmos fáciles de escribir, verificar, leer y modificar.
Los algoritmos desarrollados aplicando este modelo son más fáciles de “seguir” dado que los mismos no implementan el uso desaltos: “ir de una línea a otra línea de algoritmo” (SALTO.)

En la siguiente representación de un algoritmo genérico se trata de mostrar la utilización del mencionado SALTO:

Línea1 Acción 1

Línea2 Acción 2

Línea3 Acción 3

Línea4 Si expresión es verdadera entonces

“ir a Línea 15” …….. SALTO
sino “continuar con línea siguiente”

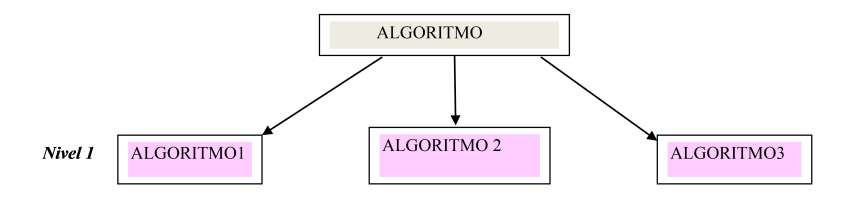
Línea5 Acción 4

Línea6 Acción 5

Línea7 Acción6

Línea15 Acción 14

Sin estos saltos, los posibles errores que pudieran existir en un algoritmo son más fáciles de detectar y corregir.La organización del algoritmo aplicando esta técnica es más evidente, dado que las acciones se encuentran agrupadas.La PEutiliza el diseño descendente o top down en el desarrollo de los algoritmos. Técnica que va de lo “general” a lo “particular”, es decir, consiste en diseñar los algoritmos en etapas, partiendo de los conceptos generales hasta llegar a los detalles.



Se podría continuar con más niveles de división. La ejecución de cada uno de los algoritmos intervinientes constituirá la ejecución del algoritmo principal.
Cada uno de estos algoritmos deberá reunir las siguientes características:

- Tener un solo punto de inicio y un solo punto de finalización.
- Toda acción de cada algoritmo es accesible, es decir, existe al menos un camino que va desde el inicio hasta el fin del mismo,que se puede seguir y pasa a través de dicha acción.
- No existirán repeticiones que nunca concluyan, es decir, infinitas.

Podríamos concluir: **la producción de un algoritmo se realizará aplicando TOP – DOWN** y además:

*“**Un algoritmo podrá ser escrito utilizando únicamente tres tipos de estructuras u organización de las acciones: secuencial, selectiva y repetitiva**”*.

La conclusión anterior es la condición fundamental de la **PE.**

2.3 Organización secuencial

La llamamos así y no estructura de control secuencial porque entendemos que esta forma de instaurar las acciones no realiza ninguna vigilancia entre las mismas, simplemente es un conjunto de acciones ordenadas de tal manera que cadaacción determina cual es la siguiente acción.

acción 1
acción 2
acción 3
———
acción n

Ejemplo:
Desarrollar en pseudocódigo el algoritmo correspondiente al siguiente enunciado:

Dados dos números reales calcular su suma y su producto y mostrar los resultados.

Algoritmo suma
variables
real: n1, n2, suma, producto
inicio
    Leer (n1,n2)
    suma ← n1 + n2
    *producto ← n1 * n2*
    Escribir 'suma =', suma, 'Producto =', producto
fin

Se puede decir que en este algoritmo las acciones Leer(), asignación de la variable suma, asignación de la variable producto y la acción Escribir() están en serie o secuencia. Es decir, cuando termina de ejecutarse una se continúa con la ejecución de la siguiente.

Actividades Propuestas

Desarrollar la estrategia solución correspondiente a cada uno de los siguientes enunciados y el algoritmo en pseudocódigo de los siguientes problemas.

1. Se tiene como datos los sueldos de tres empleados: Suel1, Suel2, Suel3 y tres descuentos variables expresados como porcentajes: Porc1, Porc2, Porc3, respectivamente. Calcular y mostrar cada uno de los sueldos netos.
2. Teniendo como datos N1 monedas de un peso, N2 monedas de cincuenta centavos, N3 monedas de veinticinco centavos, N4 monedas de diez centavos y N5 monedas de cinco centavos, calcular y mostrar cuantos pesos reúnen la totalidad de las monedas.
3. Teniendo como dato el radio de un círculo, calcular y mostrar, expresado en centímetros:
   - a) La longitud de la circunferencia que tiene dicho radio.
   - b) El área del círculo correspondiente al radio.
   - c) El volumen de la esfera correspondiente al radio.

   Además se dispone como dato del valor de 1cm en otra unidad de medida de longitud, sea del sistema métrico u otro y del nombre de la nueva unidad. Se pide calcular y mostrar los resultados obtenidos en los puntos a, b y c convertidos a la nueva unidad, identificando correctamente los resultados.

4. Tres personas deciden invertir su dinero para formar una empresa. Cada una de ellas invierte una cantidad distinta. Calcular y mostrar el porcentaje que cada una invierte con respecto al total de la inversión.
5. La serie numérica de Fibonacci declara que cada término de la misma es la suma de los dos términos anteriores. Teniendo como datos el término N1 y el posterior N2, calcular y mostrar los 5 términos siguientes.
6. Por medio del uso de una variable y sus posteriores modificaciones aritméticas, generar y exhibir los primeros cinco números naturales.
7. Ingresar un número entero N, calcular y mostrar la sumatoria de los 5 enteros posteriores a N.
8. Teniendo como dato un número par N, calcular y mostrar los cinco impares siguientes a él.
9. Dado el log10 X = 4, desarrollar un algoritmo de cálculo del valor de X.
10. Dado un número entero N, calcular y mostrar el módulo o resto entre N y 5.

2.4 Organización selectiva o decisión

Algunas veces es necesario establecer una condición en la resolución de un problema. Esa condición después de ser evaluada divide la resolución del problema en dos posibles vías o caminos: si el resultado de la condición es verdadero, se toma el camino del verdadero, si dicho resultado es falso, se toma el camino del falso.En nuestros algoritmos la condición la expresaremos mediante la construcción de una expresión algorítmica válida.

2.4.1 Estructura de selección simple: Si-entonces-sino

Este tipo de estructura organiza la ejecución de las acciones después de la evaluación de una expresión.Los resultados posibles de la expresión establecida podrán ser: verdadero (V) ó falso (F), por lo tanto las acciones siguientes a ser ejecutadas tienen dos posibles caminos. Si la expresión es verdadera se ejecutarán la/s acciones que están en el camino del verdadero, si la expresión es falsa se ejecutarán la/s acciones que están en el camino del falso.

si (expresión) entonces
    accion/es *camino del verdadero*
sino
    otras accion/es *camino del falso*
finsi

Donde **expresión** es una expresión lógica algorítmica válida y/o una variable lógica.
La cláusula **sino** puede faltar dependiendo de la condición planteada, en ese caso es una selección simple incompleta.
El margen que se establece para el **entonces** y el **sino** es para que el algoritmo sea más legible.

Ejemplos:

1.- Dados dos números reales distintos mostrar el mayor

Algoritmo numero
Variables
    real: n1, n2
inicio
Leer (n1, n2)
    si*(n1 > n2)* ***entonces***

Escribir n1
sino
Escribir n2 ***!con cláusula sino***
finsi
fin

La organización o estructura **si - entonces - sino** en el algoritmo precedente implementa dos caminos posibles dependiendo cual camino se tome según los valores de n1 y n2. Cuando n1 es mayor a n2 se exhibirá el valor de la variable n1, en caso contrario, el valor de n2. No se contempla la posibilidad n1 = n2, dado que esto ya está restringido por enunciado.

2.- Dada la edad de un alumno mostrar con cartel aclaratorio si la misma supera los 20 años.
Algoritmo edad
variable
entero: edad
inicio
Leer (edad)
si *(edad > 20)*
entonces *Escribir("Supera los 20 años")* ***!sin cláusula sino***
finsi
fin

En la estructura **si- entonces - sino** realizada en el algoritmo antecesor se muestra el cartel "Supera los 20 años", **si ysolo s**i el valor de la variable edad es mayor a 20. No se escribió la cláusula sino pues el enunciado sólo pedía el cartel si supera los 20 años.

Actividades Propuestas

Desarrollar la estrategia solución correspondiente a cada uno de los siguientes enunciados y el algoritmo en pseudocódigo de los siguientes problemas:

1. Dados dos números reales mostrar el mayor.
2. Dados cuatro números reales mostrar el mayor.
3. Teniendo como dato una temperatura en grados Centígrados, mostrar el estado físico del agua a esa temperatura y a 760 mm de presión atmosférica: sólido (menor ó igual a 0 C^o), líquido (mayor a 0C^oy menor ó igual a 100 C^o) y gaseoso (mayor a 100 C^o).
4. Un año es bisiesto si es divisible por 4 y no es por 100, o si es divisible por 400. Desarrolle un algoritmo donde se lea o ingrese desde la entrada estándar un valor entero correspondiente a un año cualquiera y determine si es o fue bisiesto.
5. Leer la nota de un alumno (numérica) y mostrar un mensaje diciendo si está reprobado (< 4), aprobado (>=4 y < 7), bueno (>=7 y <9), distinguido (= 9) o sobresaliente (= 10). Mostrar un mensaje si la nota es incorrecta. Este algoritmo debe hacerse de dos maneras distintas: con decisiones secuenciales y decisiones anidadas.
6. Siendo los datos tres números enteros, determinar si la suma de cualquier pareja de ellos es igual al número restante. Si se cumple esta condición, escribir "Iguales", en caso contrario escribir "Distintos".
7. Dados tres números enteros distintos determinar y mostrar el número que ocupa la posición del centro, al ordenar los mismos de menor a mayor.
8. Leer las coordenadas cartesianas (x, y) de un punto del plano y calcular e imprimir el cuadrante (I, II, III, IV) al cual pertenece el punto.
9. Se ingresa por teclado un número positivo de uno o dos dígitos (1..99).Mostrar un mensaje

indicando cuantos dígitos tiene el número ingresado.

10. Se tiene como datos 4 valores numéricos enteros distintos. Calcular y mostrar la suma del mayor número y el menor número de los 4 dados.

2.4.2 Estructura de selección múltiple: Según sea

En esta estructura las acciones se organizan de tal manera que pueden seguir distintas alternativas o casos dependiendo del valor que asume la variable seleccionada. La cantidad de alternativas que puedan plantearse es ilimitada.Si para distintos valores de la variable a seleccionar las acciones a seguir son las mismas, se puede resumir en un solo caso. Este tipo de estructura es más legible que la aplicación de selecciones simples anidadas.

según sea (variable)
 caso cte/s1: acción /s1
 caso cte/s2: acción /s2
 caso cte/sn: acción /sn
 ....
sino
 acción /es
fin según

donde: **variable** es la variable a seleccionar.
cte/sn es el valor/es posible/s que asume la variable seleccionada, también suele llamárselo etiqueta/s o rótulo/s
sino esta cláusula se ejecuta cuando la variable seleccionada no asume los valores contempladospor los casos.

Ejemplo:
Dado el número de un mes del año. mostrar la cantidad de días suponiendo que el año no es bisiesto.

Algoritmo cantdias
variables
 entero: mes
inicio
 Leer (mes)
 segúnsea *(mes)*
 caso *4, 6, 9,11: Escribir(“30 días”)*
 caso *1,3,5,7,8,10,12 : Escribir(“ 31 días”)*
 caso *2: Escribir(“28 días”)*
 sino
 Escribir (‘numero de mes equivocado’)
 finsegun
fIn

En el algoritmo anterior la variable mes es seleccionada y de acuerdo con su valor se ejecutarán 3 posibles opciones:

- Para mes valiendo 4 o 6 o 9 o 11, se mostrará el cartel “30 días”
- Para mes valiendo 1 o 3 o 5 o 7 o 8 o 10 o 12, se mostrará el cartel “31 días”
- Para mes valiendo 2, se mostrará el cartel “28 días”

Actividades propuestas

Desarrollar la estrategia solución correspondiente a cada uno de los siguientes enunciados y el algoritmo en pseudocódigo de los siguientes problemas:

1. Ingresar un número válido de día. Mostrar el nombre del día de la semana correspondiente.
2. Siendo los datos tres enteros válidos que representan una fecha: dd, mm, aa, se pide obtener y mostrar la fecha del día siguiente.
3. Ingresar una cadena de caracteres. Calcular y mostrar la cantidad de vocales presentes en el mismo.
4. Ingresar duplas de valores formadas cada una de ellas por un carácter y un dígito. Este ingreso no debe seguir ningún orden y no debe exceder las cuatro duplas. El carácter pude asumir los siguiente valores U: unidades, D: decenas, C: centenas, M: unidades de mil. Calcular y mostrar el número correspondiente.
5. Ingresar un número de mes válido y un año. Mostrar la cantidad de días que puede tener el mismo, considerando que el año puede ser bisiesto.

2.5 Organizacion repetitiva o iteración

Estas estructuras permiten controlar la repetición de las acciones. Las mismas se pueden repetir una cantidad conocida de veces o una cantidad desconocida de veces. Son llamadas también estructuras de iteración.

2.5.1 Estructura de Iteración con cantidad conocida de veces: Repetir Para

Esta estructura permite controlar la repetición de las acciones una cantidad predeterminada de veces. Cantidad que puede ser un valor constante o variable.

Repetir Para <variable control> ← <valor inicial>, <valor final>, <incremento>
    acción 1
    acción 2
    ———-
    acción n
fin para

Donde **variable de control o índice,** es el nombre válido de una variable perteneciente a un tipo de dato (entero). Los valores que puede asumir la variable índice están determinados por el salto o escalón entre un valor y otro de la misma. Si el salto o escalón no figura, se asume como la unidad. La secuencia de valores de la variable índice puede ser en secuencia ascendente o descendente.**valor inicial, valor final** pueden ser: un valor constante, una variable o una expresión válida.**incremento:** es el salto o escalón que asume la variable de control entre un valor y el siguiente.
El valor de la variable de control no puede ser modificado dentro del bucle o iteración, pero sí puede ser utilizado, siempre que no se lo modifique

Ejemplo:

Mostrar en secuencia ascendente los números desde el 1 hasta el 20.
Algoritmo numero
variables
entero: i

```
inicio
    Repetir Para i ← 1, 20
        Escribir (‘número = ’, i)
    finpara
fin
```

La variable i toma, sucesivamente, los valores 1, 2, 3, 4,.., 20 , siendo 1 el incremento que va sufriendo. Los valores de i serán mostrados cada vez.

2.5.2 Estructura de Iteración con cantidad desconocida de veces: Repetir Mientras

Repite le ejecución de una o varias acciones siempre que la **expresión** sea verdadera. La expresión formula una condición acerca de la resolución del algoritmo. La expresión se expresa en función de variables y/o constantes que deben tener un valor definido en el momento de su evaluación.

```
Repetir mientras (expresión) hacer
    acción/es
fin mientras
```

Donde **expresión** es una expresión algorítmica lógica válida y/o una variable de tipo lógica.
En la estructura **Repetir mientras** si la primera vez que se evalúa la expresión es falsa, la estructura mientras **NO SE EJECUTA** La/ variables que se utilizan en la expresión del mientras deben tener un valor definido antes de evaluarla y dichas variable/es deben modificar su valor dentro de la iteración, de lo contrario la misma puede quedar en un bucle infinito o loop.
Ejemplo:
Ingresar números y sumarlos, siempre que el número sea distinto de 1000

```
Algoritmo Sumar_1000_números
variables
real: num, suma
inicio
    sum ← 0
    Leer(num)
    Repetir mientras (num < >1000) hacer
        sum ← sum + num
        Leer( num )
    fin mientras
    Escribir (“La suma es ”, suma)
Fin
```

En cada repetición la variable sum se incrementa en el valor del número conocido o ingresado previamente. La variable sum deja de incrementarse cuando el número leído es 1000, el cual no se suma.En la estructura **Repetir mientras** implementada en el ejemplo, **la expresión es: num < > 1000**Llamaremos:

- ***Dato centinela:*** num (justamente es el dato que controla la repetición)

- ***Fin de Dato:*** *num = 1000 (*con este valor de num termina la repeticion)

2.5.3 Estructura de Iteración con cantidad desconocida de veces: Repetir – hasta

En este tipo de estructura la expresión también debe ser una expresión algorítmica válida o variable de tipo lógica. El cuerpo del repetir – hasta se forma con la/s acciones que deban repetirse o iterarse.Dado que la expresión se evalua al menos una vez, al final de la misma dicha acción o acciones pueden **repetir su ejecución una o más veces.** Acá igualmente la/s variables que intervienen en la expresión deben modifcar su valor para poder concluir con este tipo de estructura.

Repetir
  acción 1
  acción 2
  ....
  acción n
hasta que expresión

Ejemplo:
Ingresar números y sumarlos, siempre que el número sea distinto de 1000.

Algoritmo sumar
variables
real: num, suma
  inicio
  suma ← 0
  Leer (num)
  Repetir
    sum ← sum + num
    Leer(num)
  ***hastaque**(num = 1000)*
  Escribir (sum)
Fin

La estructura **Repetir hasta** implementada en el ejemplo anterior suma números leídos previamente hasta que el número ingresado sea 1000, al cual no suma.

Llamaremos:

- ***Dato centinela:*** num
- ***Fin de dato:*** num=1000

Actividades Propuestas

Desarrollar la estrategia solución correspondiente a cada uno de los siguientes enunciados y el algoritmo en pseudocódigo de los siguientes problemas.

1. Calcular el siguiente sumatorio. Solicitar el valor de N al usuario.

$$S=\sum \frac{1}{2^n}$$

2. Teniendo como datos dos valores enteros a y b, calcular y mostrar el producto de los mismos desarrollando un algoritmo que calcule a dicho producto como una suma reiterada.

3. Teniendo como datos dos enteros a y b, calcular y mostrar el cociente entre a y b, con b distinto de cero, como una resta reiterada.
4. Teniendo como datos dos enteros a y b, calcular y mostrar:

$$a^5$$

5. Los datos son: un número entero y la posición de un dígito del mismo, es decir posición1: unidades, posición 2: decenas, posición 3 centenas, etc, etc, etc. Mostrar el número y el dígito que se encuentra en la posición ingresada. Proponer un fin de datos.
   Ejemplo:
   Numero= 10345 y posición = 4 el algoritmo deberá mostrar: 10345 - 0
6. Se llaman números amigos a aquellos números donde cada uno de ellos es igual a la suma de los divisores del otro. Ingresar un valor entero N y mostrar todas las parejas de números amigos menores que N.
7. Ingresar un número entero N y su dígito verificador. Calcular e informar si el número ingresado es correcto calculando su dígito verificador por medio del método MODULO 11.
8. Los datos son valores numéricos enteros. Calcular y mostrar los tres valores mayores ingresados.
9. Un numero es perfecto cuando es igual a la suma de todos sus números divisores
   Ejemplo 6=1 + 2 +3
   Ingresar un número entero e informar si es perfecto.
10. Dadas las medidas de largo y alto de varios rectángulos, determinar si cada uno de ellos tiene relación áurica entre sus medidas. Proponer un fin de dato

3 — Descripción de las estructuras selectivas y repetitivas

José Eder Guzmán Mendoza - Pedro Cardona S.- Jaime Muñoz Arteaga

*Estas máquinas no tienen sentido común; todavía no han aprendido a pensar, sólo*hacen exactamente lo que se les ordena, ni más ni menos. Este hecho es el concepto más difícil de entender la primera vez que se utiliza un ordenador.

Donald Knuth

El flujo de ejecución de las líneas de un algoritmo está determinado por la complejidad de la tarea que resuelve. Un algoritmo rara vez sigue una secuencia lineal, en general evalúa una o más situaciones y responde de manera apropiada a cada una de ellas. Las estructuras de control selectivas y las repetitivas constituyen la herramienta a usar en estos casos.
Para el aprendizaje de su funcionamiento se requiere un proceso gradual de adaptación a la abstracción que representan, además de la experiencia que se vaya adquiriendo en el contexto de un lenguaje de programación en particular.
este capítulo se hace una breve referencia al concepto, utilidad y sintaxis de las estructuras de control mencionadas.
En el caso de la selectiva se desarrolla la explicación en grado de complejidad, es decir, cuando se define un bloque de instrucciones al cumplirse una condición, cuando se definen dos bloques de instrucciones y cuando se definen "n" número de bloques para "n" posibles opciones.
En el caso de la repetitiva se revisaron los diferentes tipos de ciclos que están enfocados principalmente a ciclos de tamaño relativamente fijos y ciclos que están más dependientes de variables, esto es de manera estructural porque finalmente son muy similares los diferentes tipos de ciclos.

3.1 Estructuras selectivas

Las estructuras de decisión permiten controlar el flujo de secuencia de una solución en un programa, de tal manera que en función de una condición o el valor de una variable, se puede desviar la secuencia entre diferentes alternativas.
A las Estructuras de control de Decisión también se les conoce como Estructuras de Selección.

3.1.1 Estructura de control de Selección Simple SI

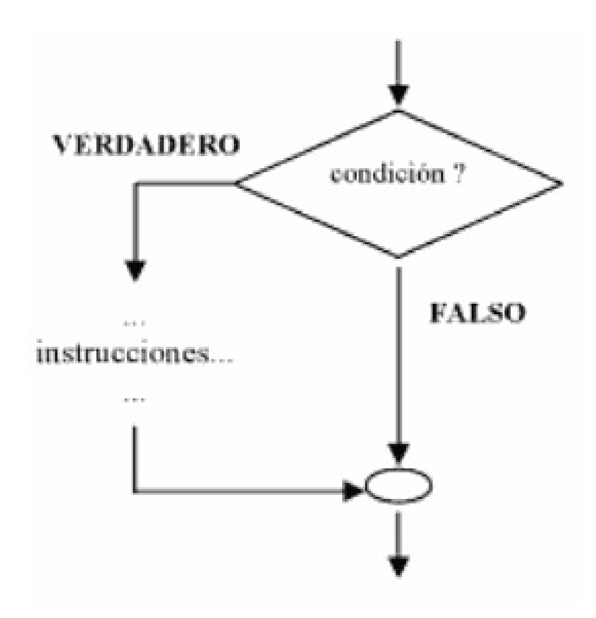
Una estructura de decisión simple "SI... FINSI", permite alterar el flujo de secuencia de un algoritmo ejecutando un conjunto de instrucciones adicionales si el resultado de una condicion es verdadera.

El formato para diseñar un algoritmo es el siguiente:

SI (Condición)
ENTONCES
instrucciones
FINSI

Explicación:
Si la condición resulta verdadera, se ejecutaran todas las instrucciones que se encuentran entre el ENTONCES y FINSI.
Si la condición resulta falsa, no se ejecuntan las instrucciones entre el ENTONCES y FINSI.

Diagrama de Flujo:



3.1.2 Estructura de control de Selección Doble SI...SINO

Una estructura de decisión doble "SI... SINO... FINSI" permite alterar el flujo de secuencia de un algoritmo ejecutando un conjunto de instrucciones adicionales dependiendo del resultado de una condición. Si la condición es verdadera, se ejecutan una serie de instrucciones, y si resulta falsa, se ejecutan otra serie de instrucciones diferentes. En esta estructura no se pueden ejecutar ambos casos a la vez, es decir, son excluyentes.

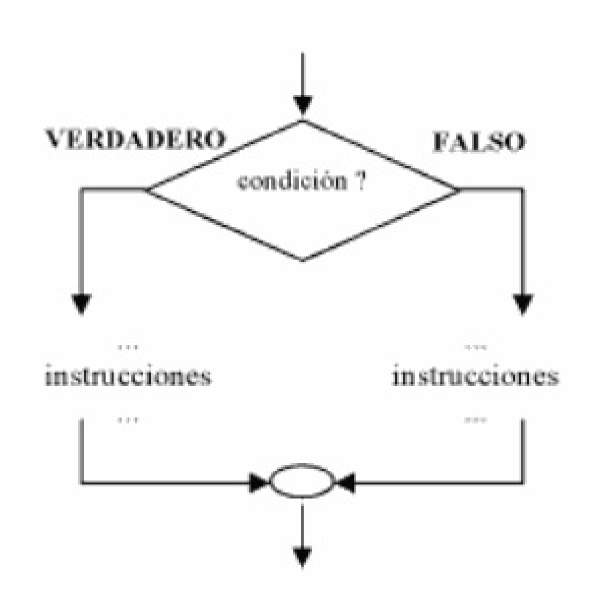
Esta estructura de decisión es de utilidad cuando la situación que se va a solucionar requiere evaluar cuál de dos posibles soluciones se va a aplicar.

El formato para diseñar un algoritmo es el siguiente:
SI (Condición)

ENTONCES
Instrucciones
SINO
Instrucciones
FINSI

Explicación:
Si la condición resulta verdadera, se ejecutan todas las instrucciones que se encuentran entre el ENTONCES y SINO.
Si la condición resulta falsa, se ejecutan las instrucciones entre el SINO y FINSI.

Diagrama de Flujo:



3.1.3 Estructura de control de Selección Múltiple

Hasta ahora hemos revisado la estructura de selección que evalúa una condición y realizan una acción en caso que se cumpla la condición; la estructura de selección que dependiendo de la condición, hace una acción si es verdadera y otra acción cuando es falsa.

Pero como vamos a dar una solución a una tarea cuando una condición puede tener múltiples opciones, es decir, que la condición que se evalúa no sea booleana (verdadero o falso). sino que es una variable que puede tener múltiples valores.

Para evaluar condiciones que pueden tener más de dos opciones se tiene la estructura de selección de decisión múltiple "CUANDO... FINCUANDO" que permite alterar la secuencia de un flujo en un algortimo ejecutando un bloque de instrucciones que dependen del valor de una variable. Para esta estructura, generalmente para cada valor posible que pueda adquirir la variable se define un bloque de instrucciones a ser ejecutadas. Además, se recomienda definir un bloque de instrucciones a ejecutar para el caso de que ninguno de los valores de la variable tenga asociado un bloque de instrucciones.

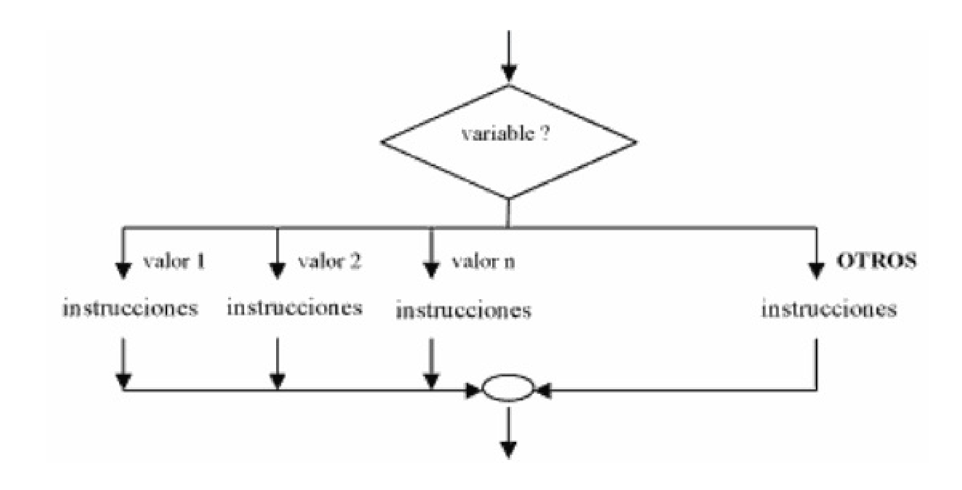
Esta estructura de selección es de utilidad cuando se requiere escoger entre más de dos posibles soluciones.

El formato para diseñar un algoritmo es el siguiente:

CUANDO (Variable) **SEA**
CASO (valor 1):
Instrucciones
CASO (valor 2): *Instrucciones*
CASO (valor n):
Instrucciones
OTROS
Instrucciones
FINCUANDO

Explicación:
Si el valor de la variable es igual a uno de los valores que aparecen en algún CASO, se ejecutan las instrucciones que están dentro del mismo.
Si el valor de la variable no es igual a ninguno de los valores en algún CASO, se ejecutan las instrucciones que están dentro delo bloque OTROS.
Nota: El bloque OTROS es opcional, es decir, puede aparecer o no.

Diagrama de Flujo:



3.2 Estructuras repetitivas

Las estructuras de repetición son bloques que permiten a un algoritmo repetir determinado segmento de código, de tal forma que, en función de una condición lógica o el valor de una variable, pueda ser repetido un determinado número de veces.
A las estructuras de control de repetición también se les conoce como estructuras de iteración o bucles, (loop).

3.2.1 Estructura DESDE

La estructura para repetir "DESDE" es una estructura de control que permite repetir la ejecución de una instrucción o un bloque de instrucciones un determinado número de veces.

Para lograr tal efecto de repeticiones se usanvariables numéricas: Un valor de inicio de repeticiones que normalmente es uno, ejemplo i=1.
Un valor final de repeticiones definida como una condición, ejemplo i<=10, en esta caso el operador es <= para que la repetición se efectué desde que es 1 hasta que sea 10.
Una variable que definirá el incremento en que se efectuara la iteración, si el valor inicial es 1 y el valor final es 10 y la variable de incremento es 1, entonces, se harán 10 iteraciones, en caso de que el incremento sea de 2, entonces, se harán 5 iteraciones.

Cada uno de los parámetros valor incial, valor final y valor de incremento son definidos y midificados según la necesidad de las tareas.

El formato para diseñar un algoritmo es el siguiente:

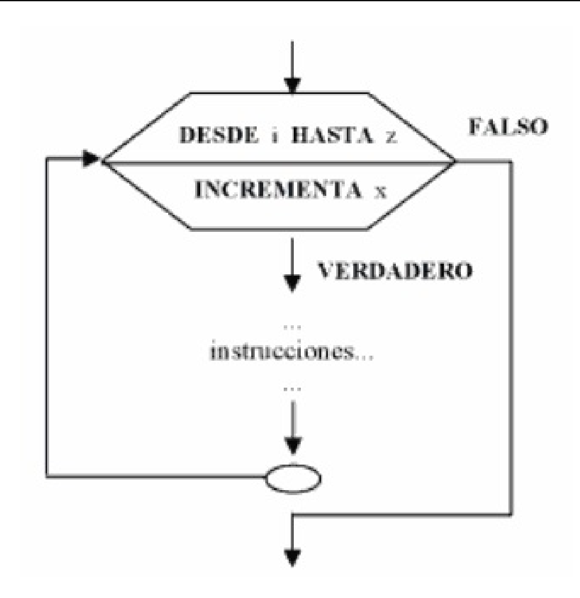
DESDE 1=valorInicial **HASTA** valorFinal **INCREMENTA** valor
Instrucción 1;
.
.
Instrucción n;
FINDESDE

La inicialización indica una variable (variable de control) que condiciona la repetición del bucle.

Repaso a las partes de la estructura DESDE:
Esta estructura tiene tres partes:

1. Inicialización: DESDE valorInicial
   Esta parte se ejecuta una sola vez al momento de iniciar la ejecución de la estructura lógica de repetición DESDE y permite asignar un valor inicial a una variable (ejemplo i=1). Esta variable funciona como un índice que permite mantener el control sobre el número de veces que se repiten las instrucciones dentro del bucle DESDE.
2. valorFinal
   Aquí se evalúa si la variable índice (valorInicial) es igual a valorFinal, es decir, si ya llegó hasta valorFinal. Si la variable índice no es igual a valorFinal, se volverá a ejecutar las instrucciones o bloques de instrucciones. En el caso de que la variable índice sea igual a valorFinal, se finalizara la ejecución del bucle DESDE.
3. Incrementa (decrementa) valor
   Cada vez que finaliza la ejecución de las instrucciones o bloque de instrucciones, se ejecuta esta parte y se incrementa (aumenta) el valor de la variable indice según el valor indicado. También existe la posibilidad de reducir el valor de la variable utilizando la regla Decrementa.

Diagrama de Flujo:



3.2.2 Estructura MIENTRAS

La estructura “MIENTRAS” permite repetir una instrucción o un bloque de instrucciones mientras que una condición se cumpla o pesta sea verdad, es decir, que el MIENTRAS se utiliza cuando realmente no sabemos el número de repeticiones que se van a realizar en el bucle, sin embargo, si sabemos que mientras se cumpla una determinada condición debemos realizar el bucle. cuando esa condición ya no se cumpla entonces el bucle finalizará.

El formato para diseñar un algortimo es el siguiente:

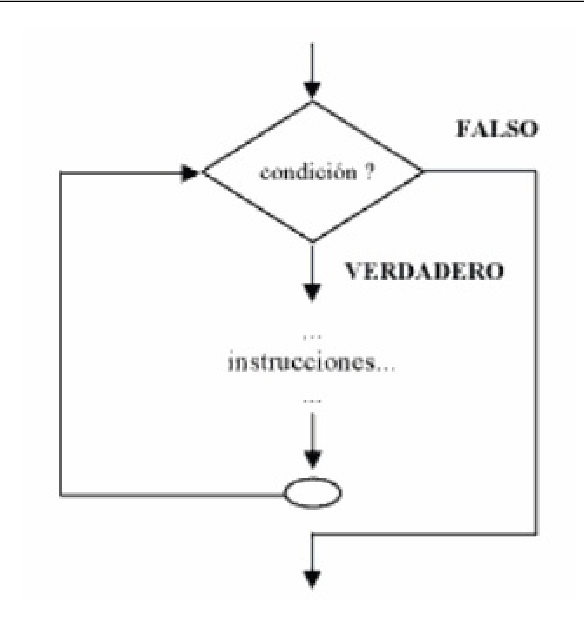
MIENTRAS (condición)
Instrucción 1;
.
,
Instrucción n;
FINMIENTRAS

Cuando se ejecuta la estructura MIENTRAS... FINMIENTRAS lo primero que se realiza es la evaluación de la condición lógica ubicada junto a la regla MIENTRAS.

Una vez que se evalúa la condición lógica, se pueden obtener dos posibles resultados:
Si el resultado es verdadero, se procede a ejecutar las instrucciones que están dentro de la estructura MIENTRAS... FINMIENTRAS y se vuelve a evaluar la condición lógica.
Si el resultado es falso, no se ejecutan las instrucciones dentro de la estructura MIENTRAS... FINMIENTRAS y se continúa con la ejecución del algoritmo.

Esto significa que mientras que la evaluación de la condición lógica sea verdadera, se ejecutará la estructura y que sólo finalizará cuando la evaluación de la condición resulte falsa.

Diagrama de Flujo:



3.2.3 Estructura HACER-MIENTRAS

La estructura “HACER-MIENTRAS” es similar a la estructura MIENTRAS, pero en el MIENTRAS la condicional lógica de repetición se evalúa al principio del bucle, y en el HACER-MIENTRAS la condición lógica de repetición se realiza al final.

Esta estructura es útil cuando se necesita repetir una instrucción o bloque de instrucciones un número de veces no determinado, siendo necesario ejecutar la instrucción o bloque de instrucciones por lo menos una vez.

El formato para diseñar un algoritmo es el siguiente:

HACER
Instrucción 1;
.
.
Instrucción n;
MIENTRAS (condición)

Cuando se ejecuta la estructura HACER-MIENTRAS se ejecutan las instrucciones que están dentro de la misma. Al final, se realiza la evaluación de la condición lógica ubicada junto a la regla MIENTRAS.

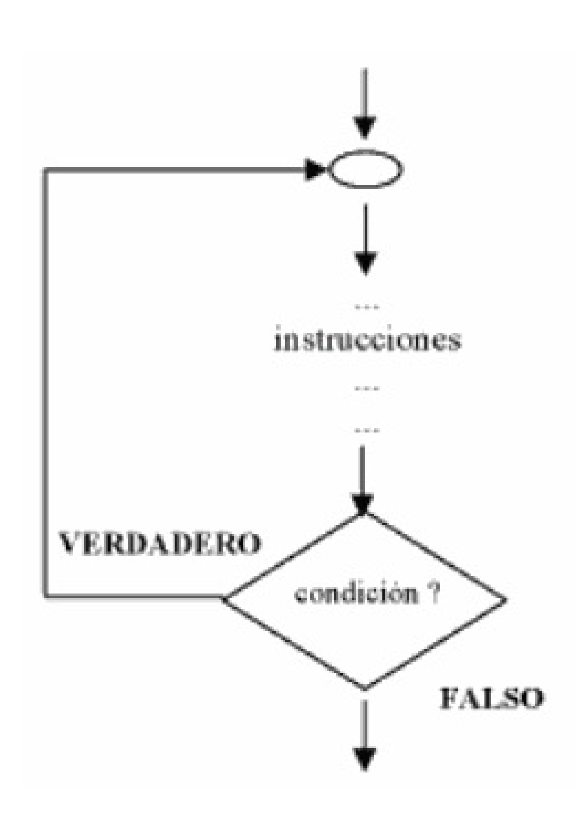
Una vez evaluada la condición lógica se pueden obtener dos posibles resultados.

Si el resultado es verdadero, se ejecutan nuevamente el bloque de instrucciones que están dentro de la estructura HACER-MIENTRAS y se evalúa nuevamente la condición lógica.

Si el resultado es falso, se continúa con la ejecución del algortimo.

Esto significa que la estructura HACER-MIENTRAS se ejecuta por lo menos una vez antes de ser evaluada la condición lógica. Si la evaluación de la condición lógica resulta verdadera, se vuelve a ejecutar la estructura, sino, finaliza la ejecución.

Diagrama de Flujo:



4 — Estructuras selectivas y repetitivas analizadas desde su uso

Edison Del Rosario Camposano

Si no puedes explicar algo de forma sencilla, entonces es que tú no lo entiendes bien.
Albert Einstein

El presente capitulo tiene como objetivo describir el uso y utilidad de las estructuras de control para elaborar algoritmos que permitan resolver tareas que son altamente asociadas con manejo de información.En un algoritmo, las estructuras de control permiten escoger el orden en que se realizaránlas acciones a seguir o repetir. Por ejemplo:

- Validar si se obtuvo la nota de suficiencia para aprobar el curso
- Permitir subir pasajeros a un transporte mientras no se sobrepase su capacidad,
- Asignar un descuento a un producto si se ha comprado mas de una docena
- entre otros.

Las estructuras de control se clasifican en dos grandes grupos:

- Estructuras de Selección: Condicionales
- Estructuras de Repetir:
  - Lazo Repita-Hasta
  - Lazo Mientras- Repita
  - Lazo Para

Los condicionales permiten seleccionar las operaciones o acciones a seguir basados en una pregunta o condición.Los lazos usan una pregunta o condición para indicar si se repite una o un grupo de acciones.

Cada una de estas estructuras son tratadas en detalle en las siguientes secciones.

En cada sección se repasan conceptos básicos para desarrollar ejercicios, que son incorporados en los algoritmos, y que serán necesarios para en las secciones siguientes.

Los diagramas de flujo para algoritmo, se realizarán con las formas básicas del estándar ISO/ANSI 5807:1985 (Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts)

Referencias

Joyanes Aguilar, L. Fundamentos de Programación. Algoritmos, estructuras de datos y objetos (2009) 3ra Edición McGraw-Hill.

4.1 Estructuras para Selección: Condicionales

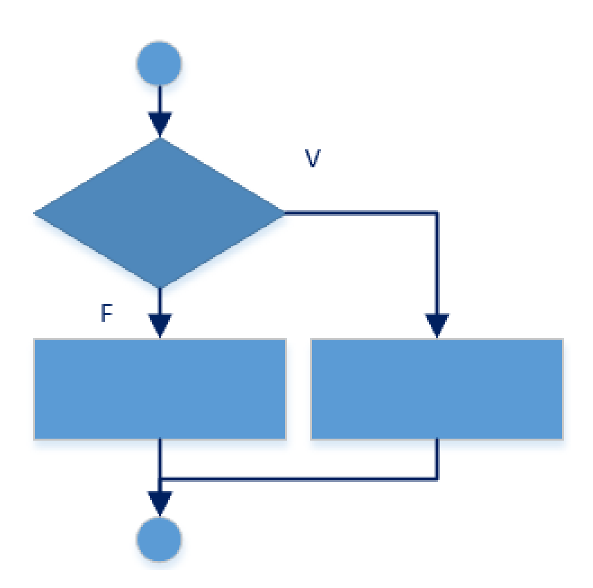
Las estructuras de selección permiten controlar el flujo de secuencia de una solución en un algoritmo, de tal manera que en función de una condicion o el valor de una variable, se puede desviar la secuencia entre diferentes alternativas.

Cuando se puede escoger entre dos caminos, se debe tomar una decisión entre las opciones.
En estos casos se utilizan los “Condicionales” que son estructuras de selección.

El condicional más simple está conformado por solo dos caminos, se formula una pregunta simple que compara al menos dos variables mediante expresiones de comparación, tales como:

a>b
a=b
a<b
a<=b
etc....

El condicional en diagramas de flujo se representa por un rombo, la expresión se escribe dentro del rombo. Por convención, hacia la derecha del rombo se grafica el caso que la expresión resultada verdadera (V), y hacia abajo o la izquierda se grafica la acción cuando el resultado es falso (F).



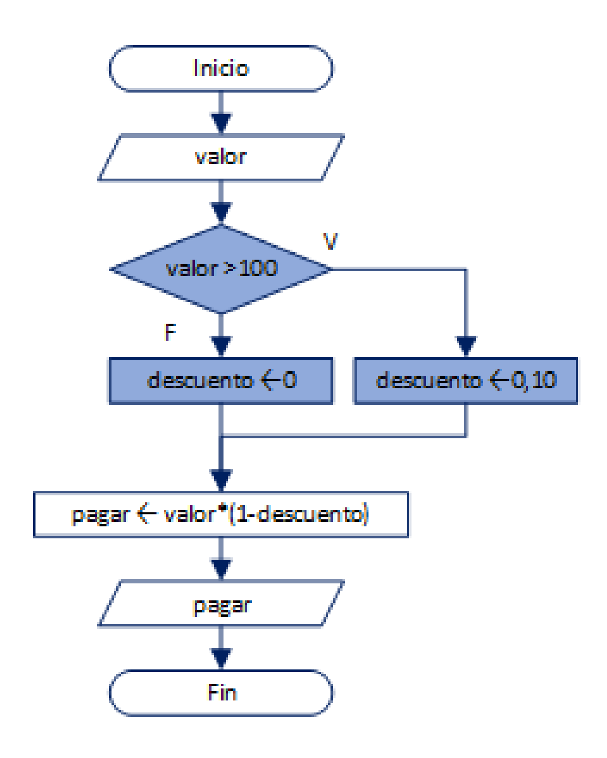
Una forma de recordar esta convención es extender la mano derecha en señal de verdad, y al extender la mano izquierda, nos queda el lado el lado falso.

Para mostrar la aplicación de un condicional en un algoritmo se utilizan los siguientes ejemplos:

Ejemplo 1.
Un almacén aplica un descuento del 10 % a quienes han comprado en productos más de 100

dólares. Realice un algoritmo que permita realizar esta operación directamente en la caja registradora.

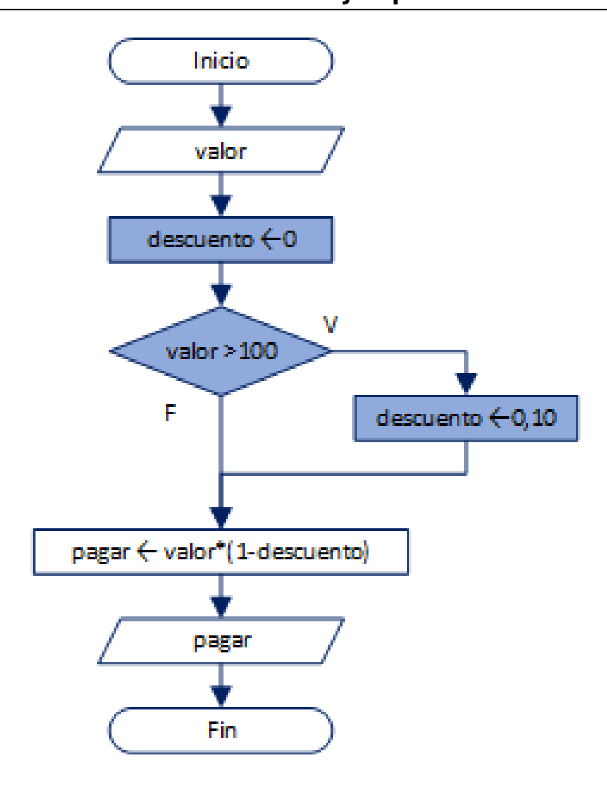
Desarrollo: El descuento lo puede aplicar el cajero, o se puede incorporar en el algoritmo de la máquina registradora, para que lo ejecute automáticamente. El **valor** de compra es la variable de ingreso, el procedimiento debe determinar el valor a **pagar**, que se presenta al final como variable de salida.



El algoritmo expresado en seudo-código:

Proceso ofertas01
Leer valor
Si valor>100 ***Entonces***
descuento←0.10
Sino
descuento←0
Fin Si
pagar←valor*(1-descuento)
Escribir pagar
FinProceso

Otra forma de expresar el algoritmo es desde el punto de vista del dueño del almacén, quien prefiere que no exista descuento (0), el cual se modifica solo si se cumple con la condición de que el cliente adquiera un valor superior a 100 dólares:



Cuando el condicional modifique una variable al cumplir solo una condición, no es necesario graficar una acción en el lado falso. En estos casos, se debe escribir la expresión de tal forma que el lado sin acción quede del lado falso.

El seudo-código para el algoritmo se presenta a continuación:

Proceso ofertas02
Leer valor
descuento<-0
Si valor>100 ***Entonces***
    descuento←0.10
Fin Si
pagar←valor*(1-descuento)
Escribir pagar
FinProceso

4.1.1 Condicional con varias preguntas

Cuando se debe incluir más de una pregunta en un condicional, se utilizan los operadores lógicos.

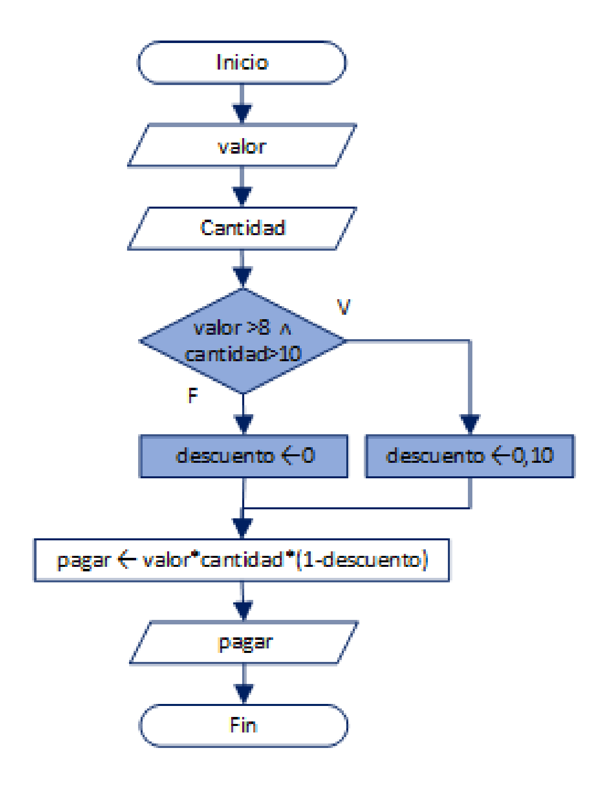
En diagramas de flujo o seudo-código, las expresiones usan las mismas conectivas y que la notación de lógica matemática. Algunos prefieren usar los la notación de lenguajes estructurados de programación u otros utilizan la forma simple: “Y”, “O”.

Conectiva	símbolo	expresión	operador
Conjunción	$\wedge$	Y	and
Disyunción	v	O	or

Un almacén aplica el descuento del 15 % para compras mayores a 8 dólares y que incluyan más de 10 artículos.

Desarrollo: Para el ejercicio se requieren dos variables de ingreso: valor y cantidad. El descuento se aplica con la expresión:
(valor $>$ 8) $\wedge$ (cantidad $>$ 10)

Para el procedimiento se continúa de igual forma que en el ejercicio anterior:



El algoritmo expresado en seudo-código:

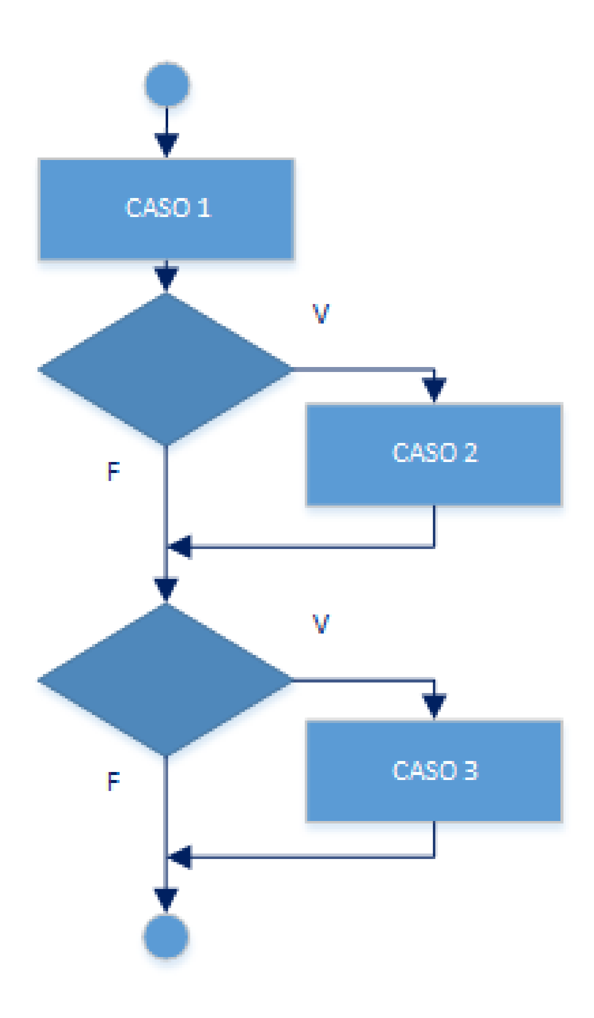
Proceso ejemplo02
Leer valor
Leer cantidad
Si (valor>8) y (cantidad>10) ***Entonces***
descuento ← 0.10
Sino
descuento ← 0
Fin Si
pagar=valor*(1-descuento)
Escribir pagar
FinProceso

4.1.2 Condicionales con casos

Existen problemas en los que una variable puede tomar diferentes valores conforme al cumplimiento de diferentes casos.

Por ejemplo cuando se realizan ventas por volumen (por cantidades grandes de producto), el descuento depende de la cantidad del producto que se vende.

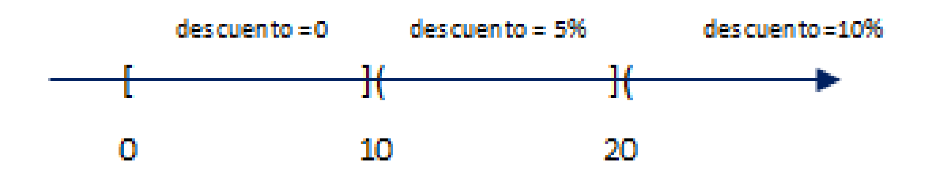
Cuando la variable resultado del condicional puede tener varios valores, se puede adoptar una forma simple del problema al dividirla por casos, empezando por el caso básico y se modifica la variable resultado lo modifica cuando se revisan los casos siguientes:

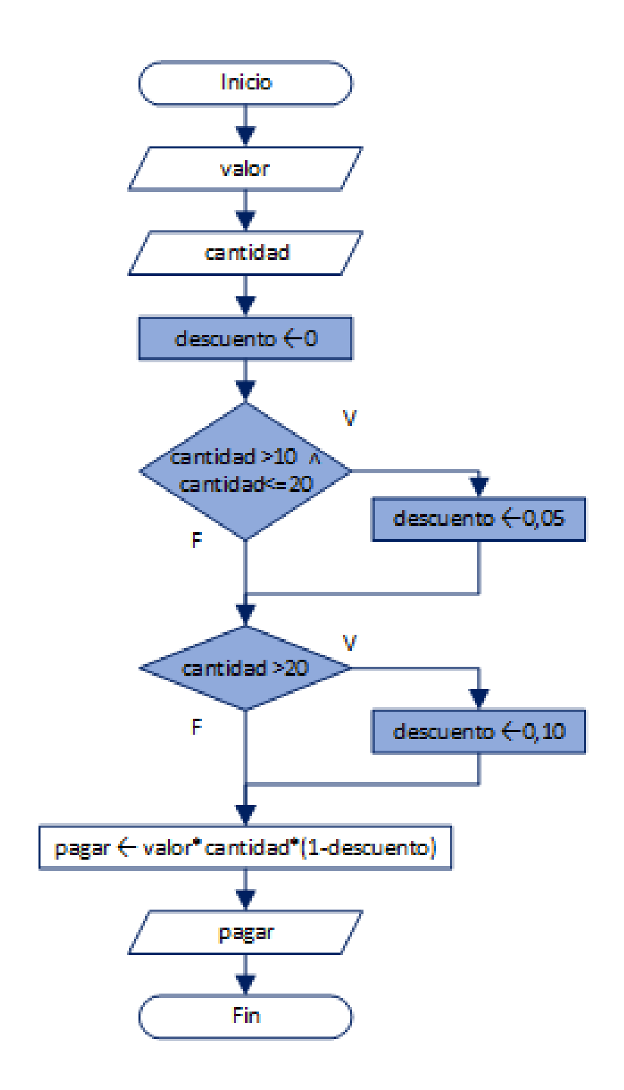


Ejemplo 3
Un almacén rebaja 10 % del precio de un producto si se adquieren más de 20 unidades y 5 % si adquieren hasta 20 unidades pero más de 10, no hay descuento para cantidades menores o iguales a 10 unidades. Con el precio unitario del producto y la cantidad adquirida, realice un algoritmo para mostrar el valor a pagar.

Desarrollo: Para realizar de forma ordenada del problema, se divide el problema en casos. El descuento se plantea de forma ascendente, representando con una recta numérica los rangos de cantidades para cada descuento. Aplicar los descuentos se traduce en evaluar cada rango de

cantidad y escribirlos uno a uno como condicionales.



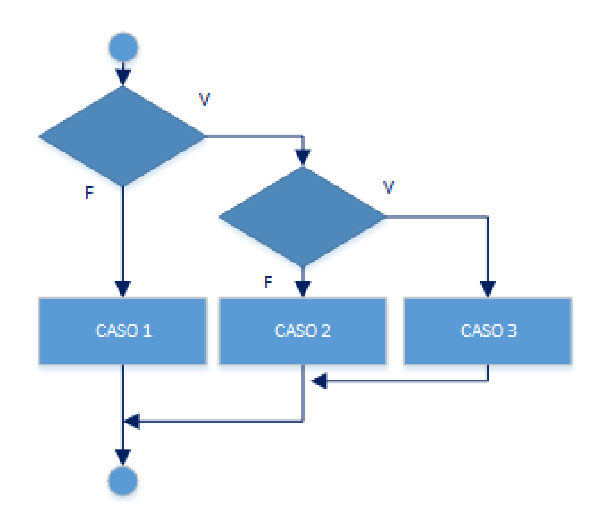


Proceso ofertas04
Leer valor
Leer cantidad
descuento<-0
Si (cantidad>10 y cantidad<=20) *Entonces*
    descuento←0.05
Fin Si
Si (cantidad>20) ***Entonces***
descuento←0.10

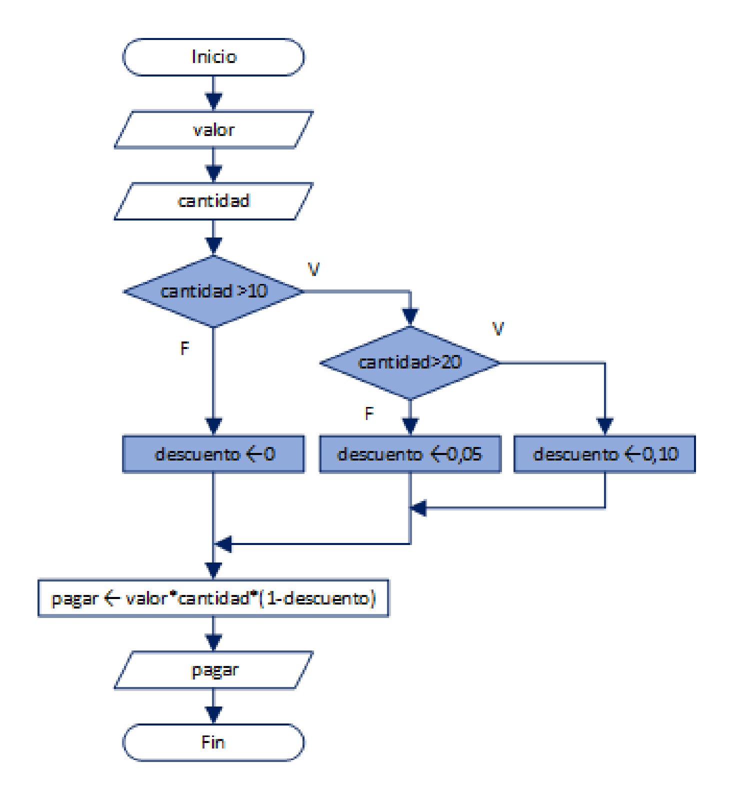
Fin Si
pagar←valor*cantidad*(1-descuento)
Escribir pagar
FinProceso

4.1.3 Condicionales en árbol

Otra forma de resolver el problema del ejemplo anterior, consiste en generar un camino por cada caso para asignar el valor a la variable de salida.
Cada camino es parte de un condicional.



usando ésta idea, el diagrama de flujo anterior se convierte en:



4.2 Estructuras para Repetir: Lazos

1. Repita - Hasta
2. Mientras - Repita
3. Lazo “Para”

Las estructuras de control de repetición son bloques que permiten a un algoritmo repetir determinado segmento de instrucciones, de tal forma que, en función de una condición lógica o el valor de una variable, pueda ser repitido un determinado número de veces.

A las estructuras de control de repetición también se les llega a conocer como estructuras de iteración o bucles.

4.2.1 Estructuras para Repetir

En algunos problemas existen bloques de operaciones que se repiten. Estas operaciones se pueden repetir hasta que se cumpla una condición o se repiten mientras se cumpla una condición.
Por ejemplo:
Un estudiante toma un curso y al final debe cumplir la condición de obtener una calificación mínima para pasar al siguiente curso, caso contrario debe repetir el curso.
Una forma diferente de expresar lo mismo es:
Un estudiante debe tomar un curso si obtiene una calificación inferior a la de suficiencia. Si la cumple, puede pasar al siguiente curso.
Como se puede observar, existen variasformas de expresar las repeticiones por lazos:

1. Repita- Hasta
2. Mientras-Repita.
3. Lazo - Para

Repita-Hasta

La primera forma de repetir mostrada es el Repita-Hasta, que ejecuta un bloque o procedimiento para luego evaluar una condición a ser satisfecha para pasar al siguiente bloque, si no se repetirá el bloque o procedimiento hasta que se cumpla la condición.

Ejemplo: un curso o materia de estudio en la universidad, al final del curso tiene la condición de superar la nota mínima para aprobar.

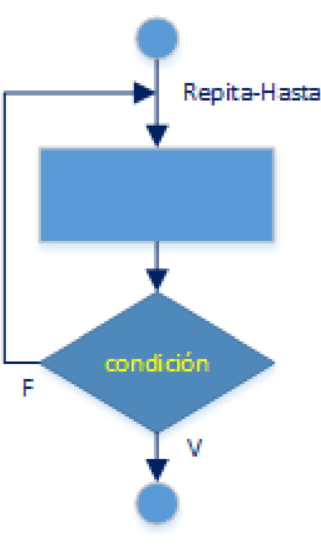


Ilustración 1

4.2.2 Mientras- Repita

Es una forma usada para cumplir una condicion antes de pasar al siguiente bloque. Si usamos la condición del punto anterior, se usará la negación de la expresión, pues para continuar al siguiente bloque se usará el lado falso.

Ejemplo: Es el procedimiento usado en las evaluaciones para nivel o cursos Inglés. El estudiante primero realiza una prueba, si no obtiene la calificación mínima debe tomar el nivel/curso A, caso contrario, puede pasar al nivel/curso B.

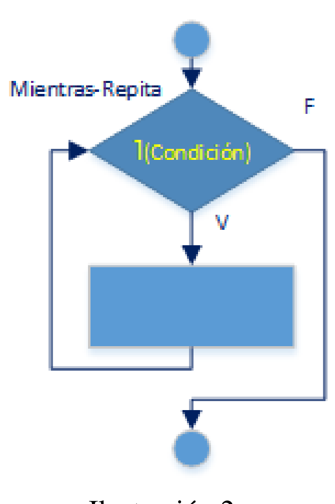


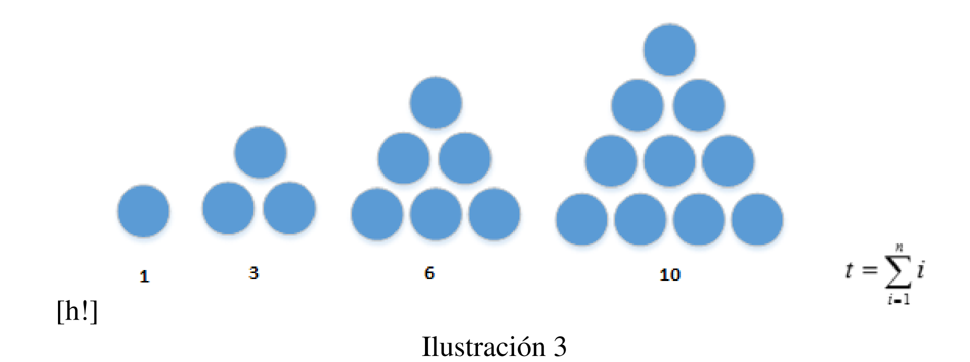
Ilustración 2

Al escribir el algoritmo se escoge el lazo "repita-hasta" o "mientras-repita" conforme la comodidad del planteamiento del problema. Sin embargo, se puede intercambiar la forma del lazo usando la negación de la expresión y cambiando la posición del condicional (inicio o al final).

Recomendación: De ser necesario para continuar, revisar la sección Conceptos y elementos básicos para la resolución algorítmica. Clasificación de Variables por uso: Contadores y Acumuladores.

Ejemplo 1

Realice un algoritmo para encontrar el término n-ésimo de la secuencia de números triangulares. Un número triangular puede entenderse como el número de elementos usados para formar una pirámide plana, como se muestra en la figura.



[h!]

Ilustración 3

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial III Término 2003 - 2004. Abril 02, 2004. Tema 2.

Desarrollo:
Usando la analogía de la construcción de una pirámide, se observa que para construir un piso se usa un bloque, para añadir el piso 2 se usan 2 bloques, para añadir el piso 3 se usan 3 bloques, etc.
Cuántos es la variable de ingreso que indica el número de pisos que se construirán.
Para iniciar los trabajos, se anuncia la construcción del primer piso y que se han usado 0 bloques.
En el procedimiento de construcción de la pirámide, **piso** es una variable tipo contador; **usados** es una variable tipo acumulador que registra los bloques **usados** en la construcción de cada **piso**.
Se construye un nuevo piso, acumulando los bloques usados hasta que se hayan construido los pisos de la variable **cuántos**.
La variable de salida que muestra el valor resultante es: **usados**.

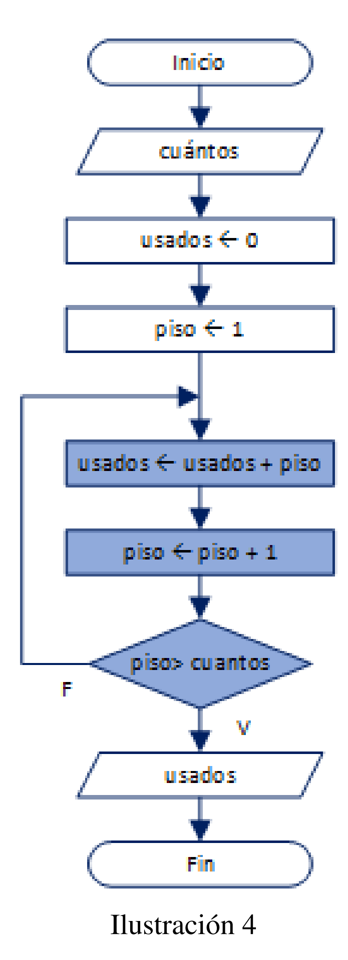


Ilustración 4

Expresar el algoritmo en pseudocódigo es:

Proceso triangular
    Leer **cuantos**
    usados ← 0
    piso ← 1
    Repetir
        usados ← **usados** + **piso**
        piso ← **piso** + 1
    Hasta Que **piso**>**cuantos**
    Escribir **usados**
FinProceso

El ejemplo se desarrolló usando el lazo Repita-Hasta, sin embargo, si se quiere cambiar el estilo usando el lazo Mientras-Repita, se lo expresa cambiando el condicional al inicio y negando la expresión:

Repita-Hasta: (pisos>cuántos)
//construir **pisos** hasta que se completen los pedidos en la variable **cuántos**
Mientras-Repita: ⌉(pisos>cuántos)
(pisos≤ cuántos)
//Construir **pisos** mientras sean menores e iguales a los pedidos en la variable **cuántos**.

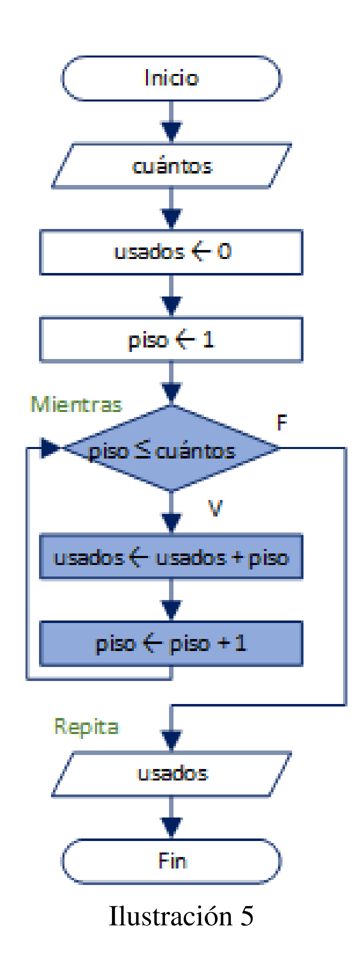


Ilustración 5

En pseudocódigo el algoritmo se escribe como:
Proceso triangular
Leer **cuantos**
usados←0
piso←1
Mientras piso<=cuantos Hacer
usados ← **usados** + **piso**
piso ← **piso**+1
Fin Mientras
Escribir **usados**
FinProceso

Ejemplo 2

Extendiendo el ejercicio anterior, construya un algoritmo para determinar si un número **t** es triangular.

Para esto el lazo se debe controlar mediante el número de bloques **usados** comparados con el número **t**.

Si los **usados** son iguales o mayores a **t**, no es necesario continuar el proceso pues se obtuvo un resultado afirmativo, o los usados superan lo requerido y el número **t** no es triangular. Puede mostrarse la respuesta como un valor verdadero (1) o falso (0) al comparar la igualdad entre **t** y **usados**.

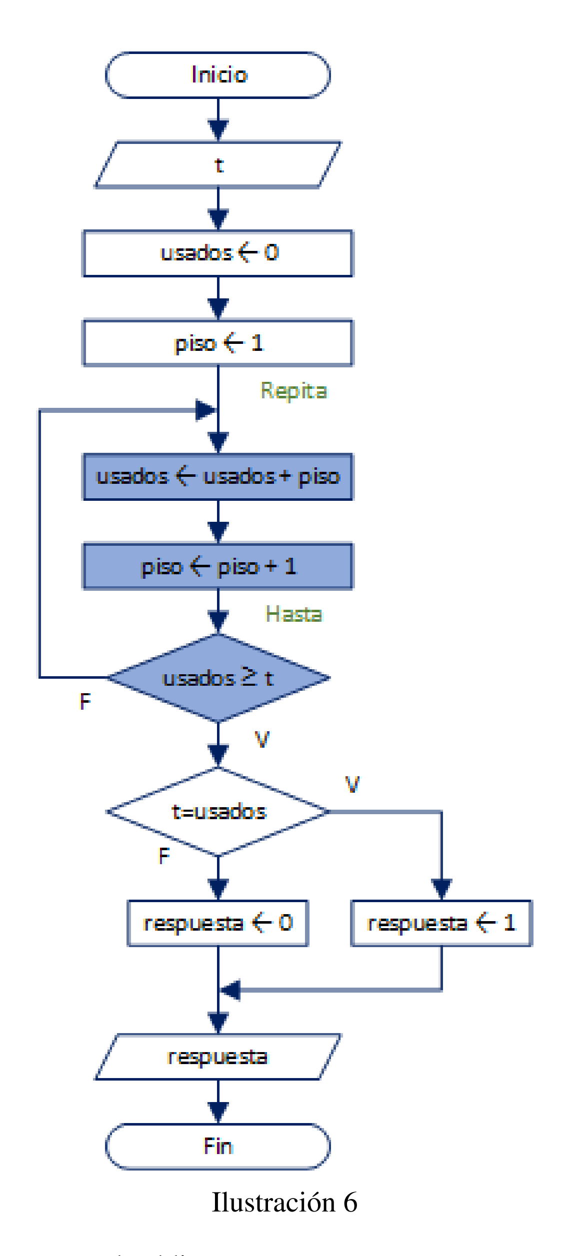


Ilustración 6

El algoritmo presentado en pseudocódigo es:

```
Proceso verificatriangular
    Leer t
        usados←0
        piso←1
    Repetir
        usados←usados+piso
        piso←piso+1
    Hasta Que usados>=t
    Si t=usados Entonces
        respuesta←1
    Sino
        respuesta←0
    Fin Si
    Escribir respuesta
FinProceso
```

Propuesta: Realizar el ejercicio usando la forma mientras-repita.

Ejemplo 3

Un entero es divisible para 9 si lo es la suma de sus cifras. Conociendo ésta propiedad, realice un algoritmo para determinar si un número **n** es divisible para 9.

Ejemplo:

n = 1492; 1+4+9+2 = 16; 1+6=7, por lo que el número no es divisible para 9
n = 1548; 1+5+4+8 = 18; 1+8=9, el número si es divisible para 9

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial II Término 2001 - 2002. Diciembre 11, 2001.Tema5.
http://es.wikipedia.org/wiki/Divisibilidad

Desarrollo:

La variable de entrada usada es **n**, que corresponde al número a evaluar.

En el procedimiento, para separar los dígitos del número **n**, se usará el residuo de la división para 10, que permite extraer cada dígito en forma consecutiva. Se acumulan los residuos en una variable **s**, y de ser necesario se repite el proceso hasta quedar con un resultado **s** de un dígito, para verificar que sea o no 9.

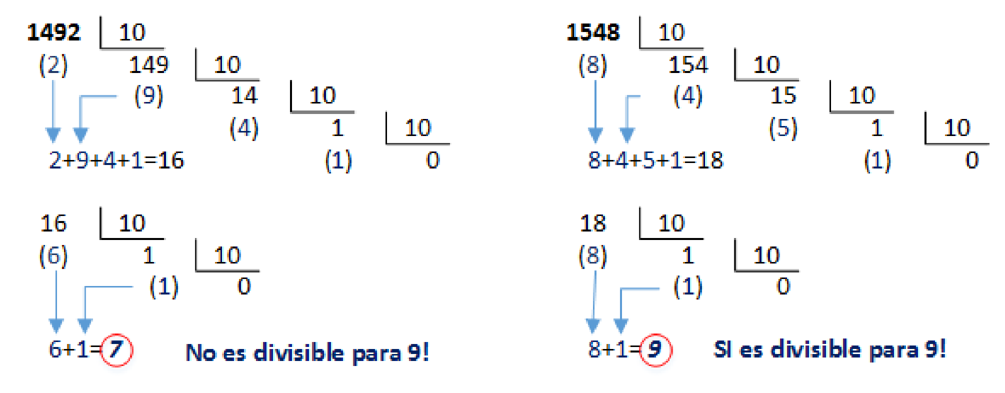


Ilustración 7

Primero hay que desarrollar el algoritmo para acumular dígitos

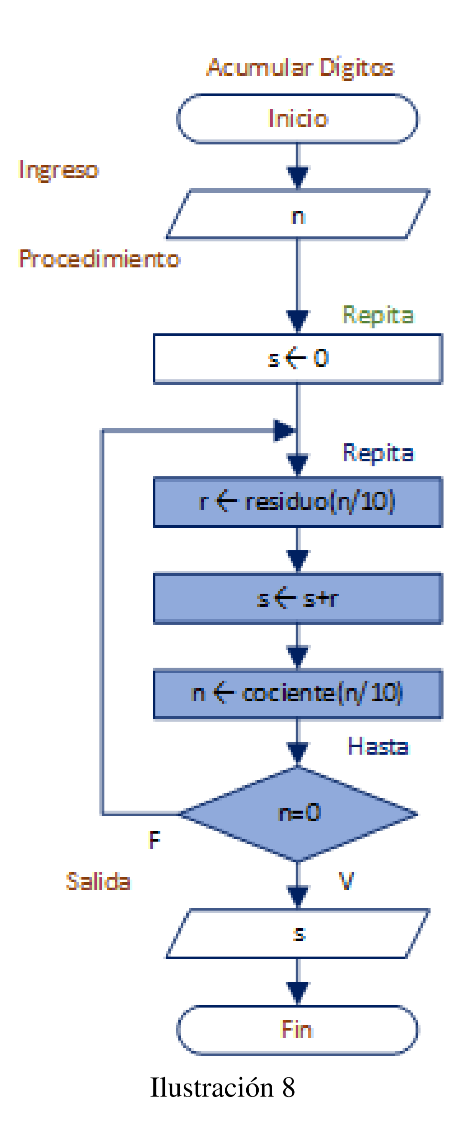


Ilustración 8

Luego se puede ampliar para abarcar la suma de dígitos hasta obtener un resultado de un dígito, que nos permitirá determinar si el número **n** es o no divisible para 9.

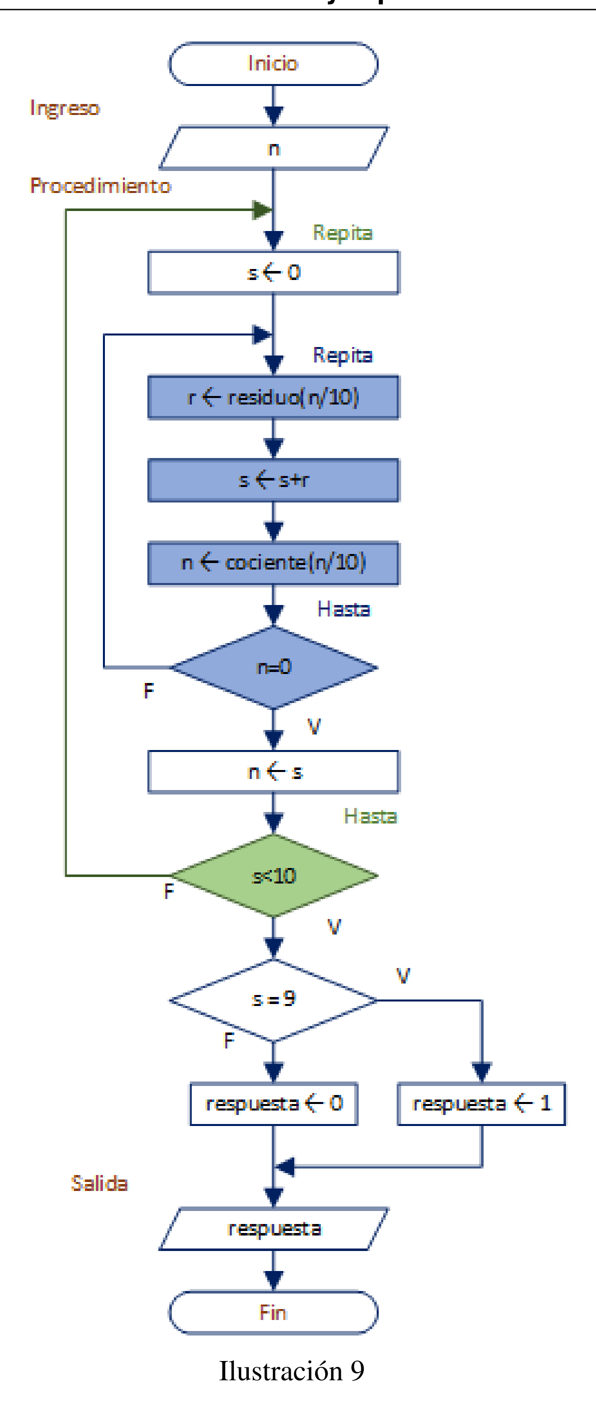


Ilustración 9

Expresar el algoritmo en pseudocódigo consiste en traducirlo a:

Proceso divisible9
Leer **n**
Repetir
    s←0
    Repetir
        r←**n** MOD 10
        s←**s**+**r**
        n←trunc(**n**/10)
    Hasta Que **n**=0

n←**s**
Hasta Que **s**<10
Si **s**=9 Entonces
respuesta←1
Sino
respuesta←0
Fin Si
Escribir **respuesta**
FinProceso

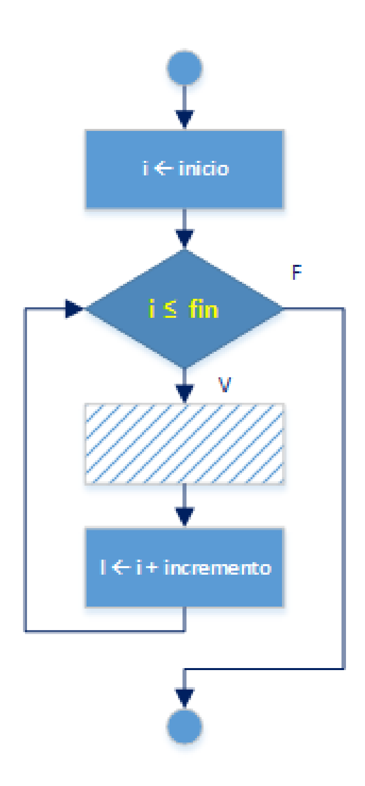
Propuesta: Desarrollar el ejercicio 3 usando lazo mientras-repita

Ejercicios.

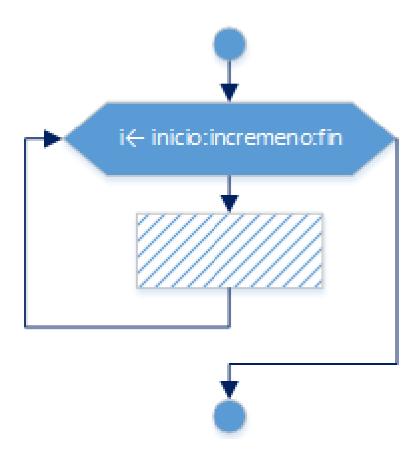
Lazo "Para"

En algoritmos es frecuente encontrar lazos controlados por solamente un contador, por lo que se pueden integrar y resumir en una forma de representación conocida como el lazo "Para".

Usamos un diagrama de flujo para mostrar el concepto con un lazo Mientras-Repita. El contador **i** con valor de **inicio** permite el control de operaciones de un bloque mediante el lazo **mientras** con la expresión i menor o igual que **fin**. Luego del bloque de operaciones, se **incrementa** el valor de **i** para revisar la condición del lazo y repetir si es necesario.



Integrando las operaciones del contador y el lazo, en un diagrama de flujo se pueden simplificar en una estructura de control conocida como "Para":



Se puede leer también: “Para los valores de **i** empezando en **inicio**, avanzando en pasos de **incremento** hasta llegar al valor de **fin**, repita el siguiente bloque de operaciones”.

Para lazos con expresiones de repetir más complejas (i>n y j<k) será más práctico usar las formas repita-hasta o mientras-repita, como se mostrará en la sección de ejemplos.

4.3 Estructuras de Control - Condicionales y Lazos

1. **Ejercicios y Aplicaciones básicas**
2. **Ejercicios de Lazos con Bases numéricas**
3. **Ejercicios de Lazos con Aleatorios**

4.3.1 Ejercicios y Aplicaciones básicas

En ésta sección se proponen algunos ejercicios y ejemplos que permiten practicar algunas aplicaciones de algoritmos siguiendo las siguientes premisas:

Los ejercicios pueden ser más complejos de lo propuesto, sin embargo el enunciado se ha simplificado para permitir al estudiante desarrollar algoritmos con un nivel inicial de complejidad.

Por lo que algunos ejercicios serán nuevamente enunciados con una complejidad mayor en capítulos posteriores cuando se disponga de estructuras/herramientas que permitan un desarrollo más completo.

Ejercicio 1

Se dice que un número de dos cifras es primo permutable si al intercambiar sus cifras sigue primo.

Ejemplos 37, 17, 19 etc.

Realice un algoritmo para determinar si un número **x** es primo permutable.

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Mejoramiento I Término

2002 - 2003. Septiembre 24, 2002. Tema 2

Ejercicio 2
Una fábrica produce botellas de vidrio reciclando botellas usadas. Suponiendo botellas similares, la maquina con x botellas usados pueden fabricar 1 botella nueva.



Realice un algoritmo para encontrar la cantidad total acumulada de botellas que pueden fabricarse a partir de **n** botellas en el mercado, reciclándolas repetidamente hasta que ya no quede suficientes botellas para reciclar.

Por ejemplo: Si n = 70, x = 4, la respuesta entregada por el algoritmo es 23 siguiendo el siguiente proceso:

Primer reciclaje: se fabrican 70/4 =17 botellas y sobran 2
Segundo reciclaje: n= 17 + 2 = 19, se fabrican 19/4 = 4 botellas y sobran 3
Tercer reciclaje: n= 4 + 3 =7, se fabrican 7/4 = 1 sobra 3
Cuarto reciclaje: n=1+3=4, se fabrican 4/4=1 botella y ya no quedan suficientes botellas para reciclar. El algoritmo termina y muestra la cantidad acumulada.
Total de botellas fabricadas: 17 + 4 + 1 +1= 23

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Final II Término 2004 - 2005. Febrero, 2005. Tema 3.

Ejercicio 3
Un número perfecto es aquel que es igual a la suma de todos sus divisores, con excepción del mismo. Realice un algoritmo que valide si un número **x** es o no perfecto, respondiendo mediante valores lógicos 1 y 0.
Ejemplo:
6 es perfecto porque sus divisores son: 1, 2, 3 (6 no se considera). Sumados 1+2+3=6

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Final II Término 2002 - 2003. Febrero 13, 2002. Tema 1

Ejercicio 4
El ISBN o "International Standard Book Number" es un número de 10 dígitos usado para identificación de libros, cuyo último dígito es un verificador que se calcula mediante una operación con los dígitos anteriores.
El dígito verificador es obtenido mediante el residuo de **S** para 11.
Donde **S** es la suma de una vez el primer dígito, más dos veces el segundo dígito, mas tres veces el tercer dígito, . . . , más nueve veces el noveno dígito.
Escriba un algoritmo que lea un número ISBN y verifique si fue escrito de forma correcta.



Ejemplo:
La suma S para el ISBN 9684443242 es:
1*9+2*6+3*8+4*4+5*4+6*4+7*3+8*2+9*4 = 178
El digito verificador es residuo(178/11) que es igual a 2.

Referencia:
ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial I Término 2004 - 2005. Julio 06, 2004. Tema 2.

Ejercicio 5
El número π puede ser obtenido mediante aproximaciones siguiendo la serie conocida como "producto de Wallis", desarrollada por matemático inglés John Wallis en el año 1655.

$$\frac{2}{1}\times\frac{2}{3}\times\frac{4}{3}\times\frac{4}{5}\times\frac{6}{5}\times\frac{6}{7}\times\frac{8}{7}\times\frac{8}{9}\times \ldots = \frac{\pi}{2}$$

Realice un algoritmo para encontrar el valor aproximado de π con la fórmula mostrada para **n** dado.

Referencia:
ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial I Término 2005 - 2006. Julio 05, 2005. Tema 2.

Ejercicio 6
La sucesión de Padovan es la secuencia de números enteros P(n) definida por los siguientes valores iniciales:
P(0)=P(1)=P(2)=1
y el valor siguiente: P(n)=P(n-2)+P(n-3).
Describa un algoritmo estructurado que calcule y muestre el término n de la sucesión, considere que n $>$3.

Ejemplo:
Los primeros valores de P(**n**) son:
1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37,...
Si **n**=15, el numero buscado es 37

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación II Término 2012-2013.Noviembre 27, 2012. Tema 1.

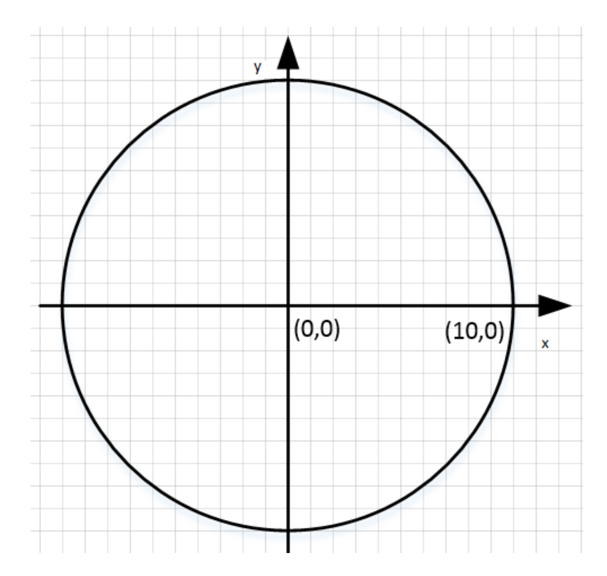
Ejercicio 7
Omirp se define como un número primo que al invertir sus dígitos da otro número primo. Escriba un algoritmo para determinar si un número n tiene la característica de ser un número Omirp.

Ejemplo:
1597 es número primo,
Se invierte sus dígitos: 7951
7951 es primo,
Entonces el número 1597 es un número omirp.

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación I Término 2010-2011. Julio 6, 2010. Tema 2.

Ejercicio 8
Escriba un algoritmo para determinar el número de puntos del plano cartesiano con coordenadas de valores enteros que pertenecen al círculo limitado por la circunferencia de ecuación x2 + y2 = 100 (centro en el origen y radio 10).

Muestre también el promedio de las distancias de dichos puntos al origen de coordenadas.



Sugerencia: Utilice un circulo inscrito en un cuadrado, para cada punto con coordenadas enteras, calcule la distancia al origen para determinar si el punto está dentro del círculo.

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial III Término 2003 - 2004. Abril 02, 2004. Tema 3

4.4 Estructuras de Control - Lazos con Bases numéricas y Aleatorios

1. **Ejercicios de Lazos con Bases numéricas**
2. **Ejercicios de Lazos con Aleatorios**

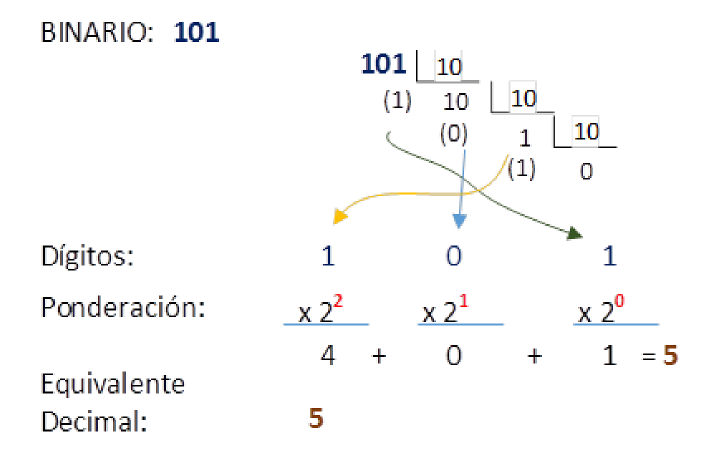
Continua de la sección anterior

4.4.1 Ejercicios de Lazos con Bases numéricas

En caso de requerir repasar el tema de bases numéricas, puede remitirse al Capitulo 8-Sistemas de numeración para la representación a Bajo Nivel

Ejemplo 1

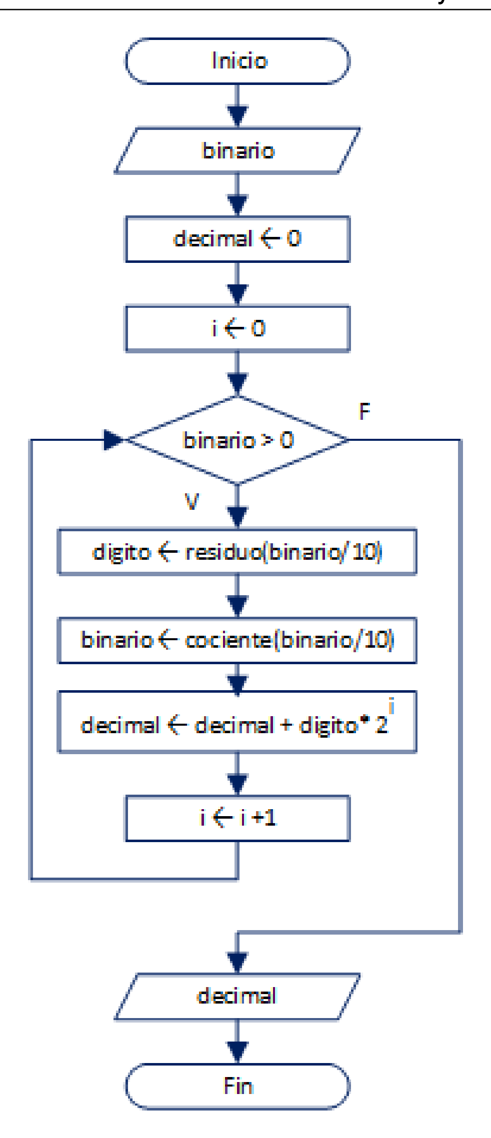
Para convertir el número binario 101 a decimal, se realizan las siguientes operaciones:



Algoritmo de Binario a Decimal

Para realizar el algoritmo, se tiene que separar cada dígito del número binario, y realizar la operación de ponderación para acumularla en el resultado final.

En la ponderación se usará un contador de posición para ser usado como el exponente de la base 2.
El algoritmo en diagrama de flujo se representará como:



Y en su forma de seudo-código:

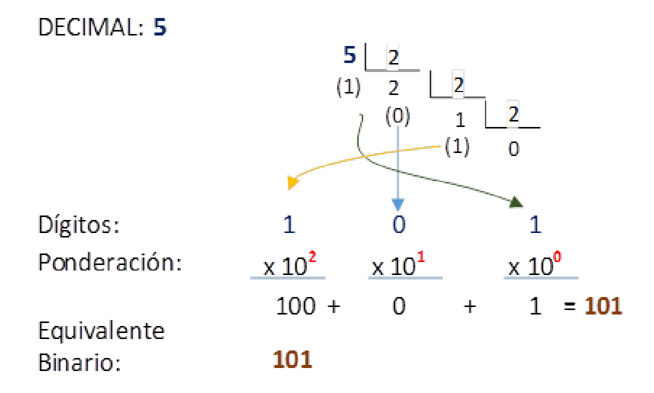
```
Proceso BinarioDecimal
    Leer binario
    decimal ← 0
    i ← 0
    Mientras binario > 0
        digito ← binario mod 10
        binario ← trunc(binario/10)
        decimal ← decimal + digito * 2
        i ← i + 1
    FinMientras
    Escribir decimal
FinProceso
```

Algoritmo de Decimal a Binario

En este caso hay que analizar descomponer el número en la nueva base numérica, para luego trabajar con los residuos ubicándolos en desde la posición menos significativa a la más significativa.

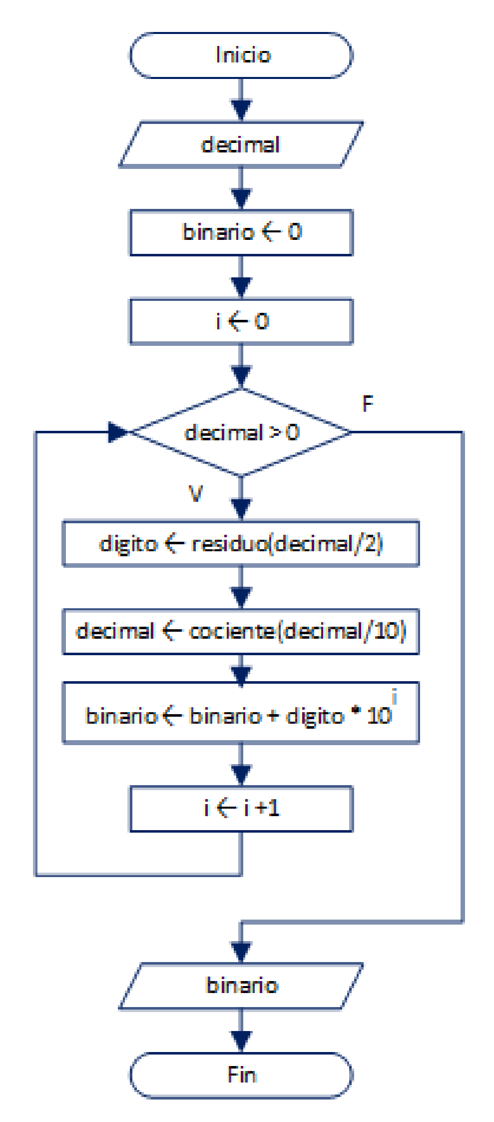
Ejemplo 2

Para convertir el número decimal 5 a binario, se realizan las siguientes operaciones:



Observe la diferencia en las operaciones para el divisor y la base, las operaciones son similares a las del ejercicio anterior. Por lo que el algoritmo no debe representar mayor inconveniente al ser una variante del ejemplo anterior.

Algoritmo Decimal a Binario



El ejemplo en seudo-código:

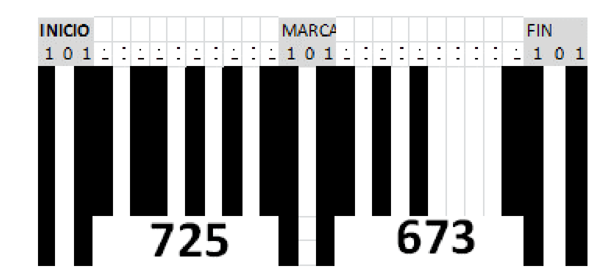
Proceso DecimalBinario
Leer decimal
    binario ← 0
    i ← 0
Mientras decimal > 0
    digito ← decimal mod 2
    decimal ←trunc(decimal/2)
    binario ← binario + digito * 10
    i ← i + 1
FinMientras
Escribir binario
FinProceso

Ejercicios:

1. Realizar los algoritmos Binario a Decimal y Decimal a Binario usando el lazo Repita-Hasta.
    a) Crear un algoritmo para convertir un número de base numérica **x** a base **y**.
    b) Comprobar el algoritmo usando los valores: 23x númeroy.
    c) Sugerencia, convierta el número de la base x primero a decimal y luego transforme el resultado a la siguiente base numérica y.
2. El código de barras utiliza líneas paralelas verticales (barras y espacios) que representan información en su equivalente binario. El código es muy usado en los puntos de ventas y es "leído" por un dispositivo láser (scanner).
   Para facilitar la lectura por scanner se usa el método de "simbología discreta", en el que se marca el inicio, separación y fin de los datos con la secuencia barra/espacio/ barra (101) por cada grupo de 10 bits (dígitos binarios).

   Elabore un algoritmo que permita cambiar un código de producto conformado por dos números de 3 cifras a su equivalente en código de barras usando simbología discreta.

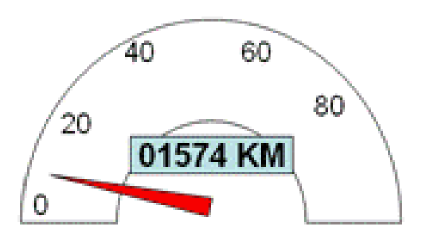
   Referencia: http://es.wikipedia.org/wiki/C%C3%B3digo_de_barras



   Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación I Término 2013-2014. Julio 2, 2013. Tema 2.

3. En un odómetro mecánico de un vehículo antiguo se marcan las distancias recorridas en kilómetros, en formato numérico octal de hasta cinco dígitos.
   Realice un algoritmo para encontrar la distancia recorrida en kilómetros en formato numérico decimal, convirtiendo el valor octal marcado por el odómetro luego de un viaje.

Nota.- Un odómetro es un dispositivo que indica la distancia recorrida en un viaje de un vehículo.



Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación II Término 2008 - 2009. Diciembre 09, 2008. Tema 1.

a) Para realizar la conversión de un número que está en una determinada base a su equivalente decimal, debe emplearse la siguiente regla:

$$N = d_iB^i + \ldots + d_4B^4 + d_3B^3 + d_2B^2 + d_1B^1 + d_0B^0$$

En donde:
B: Base del sistema de numeración original
di: dígito en la posición i, i = 0, 1, 2, ... (0 es la posición menos significativa)

a) escriba un Algoritmo que permita obtener el equivalente decimal (base 10) de un numero octal (entero de hasta 4 dígitos) ingresado por teclado. Suponga que ya existe la función EsOctal(n), cuyo parámetro n es un valor entero y retorna 2 posibles valores: 1 = verdadero, 0 = falso, según sea que n es válido o no en ese sistema de numeración.
b) Realice la prueba de escritorio del algoritmo construido en el literal a) para el siguiente ejemplo: 1034(base 8) = N(base 10)
Ejemplo:
para convertir 764(base 8) a (base 10):
N = 7 x 8^2 + 6 x 8^1 + 4 x 8^0 = 500(base10)

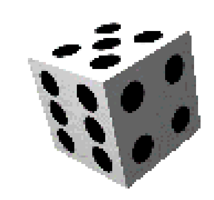
Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial II Término 2003 - 2004. Diciembre 09, 2003. Tema 1

4.4.2 Ejercicios de Lazos con Aleatorios

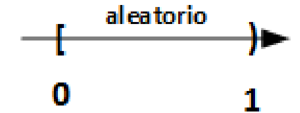
Aleatorios
Para simular un número de azar, por ejemplo el obtenido al lanzar un dado, se recurre al concepto de números aleatorios.
Un número aleatorio se define como un número cualquiera real en el rango [0,1)



Para utilizar el número aleatorio se debe convertir al rango apropiado del número a simular, por ejemplo, para simular un dado se escribiría en el algoritmo:



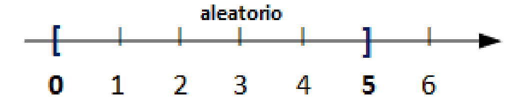
dado ← entero(aleatorio*6)+1
Que básicamente describe que:
- aleatorio es un número real [0,1),



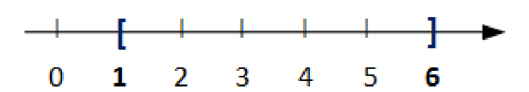
- como el dado tiene 6 caras se multiplica por 6, obteniendo un real de [0,6). No se incluye el 6.



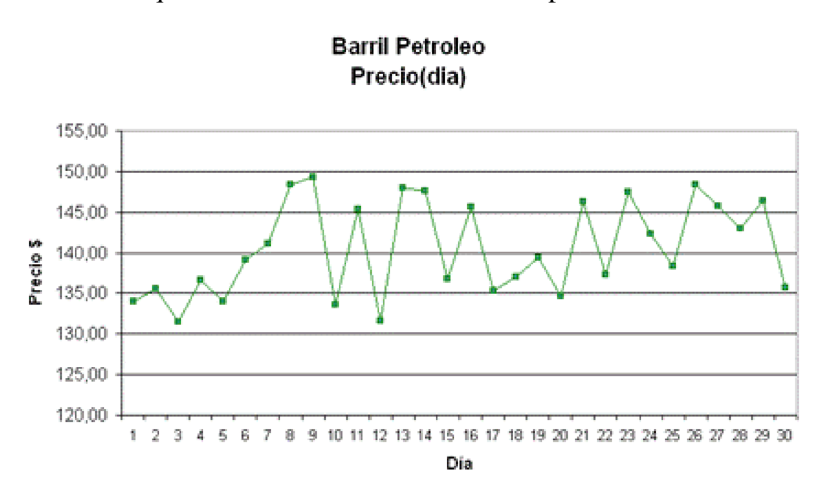
- Se extrae solo la parte entera para obtener un número entero [0,5]



- Para que el resultado sea [1,6], se le suma 1



Ejercicio 1
Realice un algoritmo para simular el precio del barril de petróleo durante un mes de 30 días, suponiendo que son valores enteros que fluctúan en forma aleatoria entre $ 130 y $ 150 y se obtenga las siguientes respuestas:
a) El promedio del precio del petróleo.
b) ¿Cuál fue el día en el que estuvo más barato el barril de petróleo?

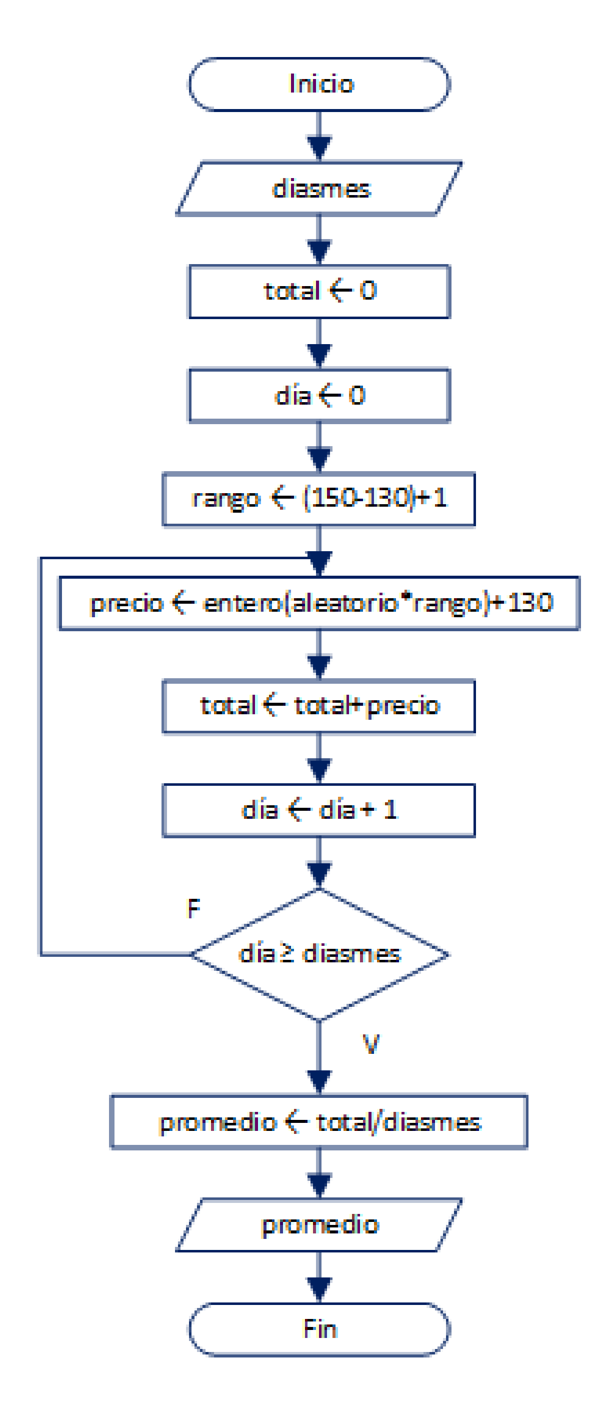


Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación I Término 2008 - 2009. Julio 08, 2008. Tema 3

Desarrollo: Para iniciar el algoritmo, se puede considerar como variable de entrada los días del mes, o asignarles directamente 30 días.

La primera aproximación al problema para responder el literal a) consiste en generar números aleatorios en el rango [130, 150] y acumular sus valores para el promedio. Será necesario disponer de un contador para controlar el número de veces que se generan los precios de forma aleatoria en el lazo de repetición.

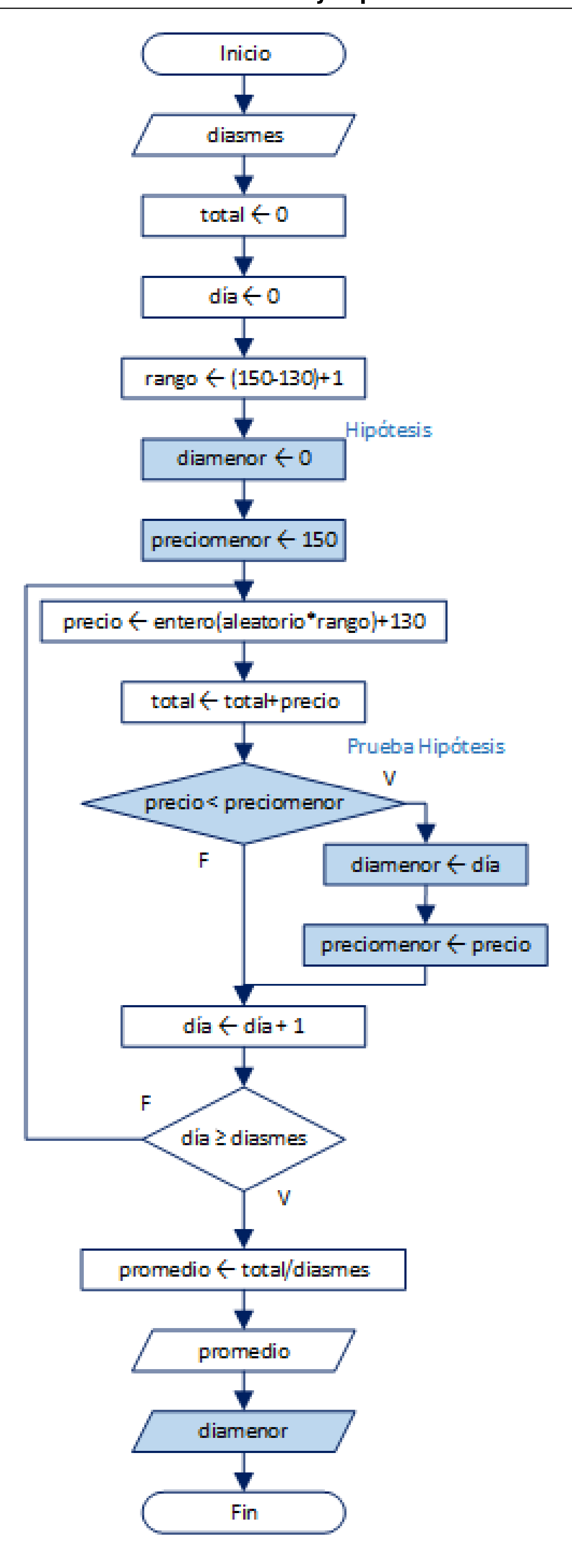
Una de las formas de resolver el problema es con un lazo repita hasta, cuyo diagrama de flujo se muestra a continuación:



```
Proceso promedioprecio

Leer diasmes
    total←0
    dia←0
    rango←(150-130)+1
Repetir
    precio←TRUNC(AZAR(rango))+130
    total←total+precio
    dia←dia+1
Hasta Que dia>=diasmes
    promedio←total/diasmes
Mostrar promedio
FinProceso
```

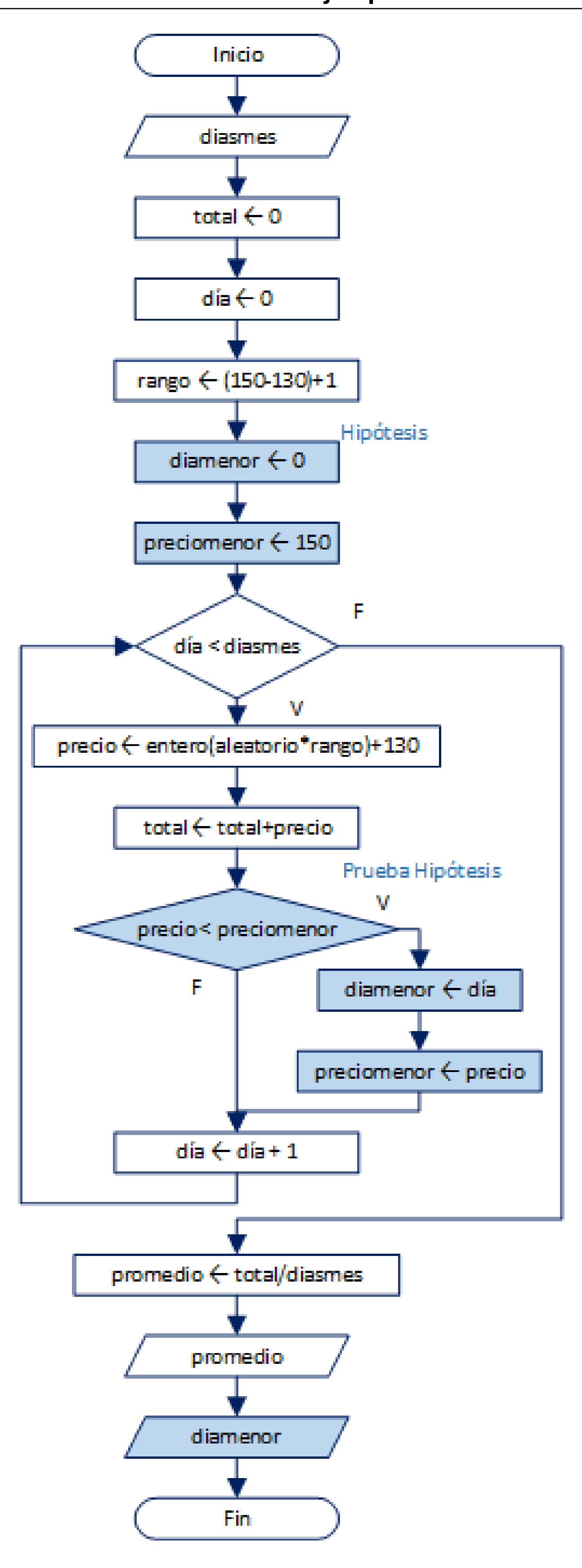
Para la pregunta b) es necesario analizar la manera de encontrar el día con el precio más barato. En este caso se utilizará el algoritmo para búsqueda del menor, que consiste en iniciar con el supuesto para el valor menor de precio y día, probando contra el precio de cada día y de ser necesario se cambian los valores menores. Es un similar al caso de usar una hipótesis y realizar luego las pruebas.Como supuesto, se escogerá el valor máximo de precio con el objetivo que el primer precio que aparece sustituye los valores.


Inicio
diasmes
total ← 0
día ← 0
rango ← (150-130)+1
Hipótesis
diamenor ← 0
preciomenor ← 150
precio ← entero(aleatorio*rango)+130
total ← total+precio
Prueba Hipótesis
V
precio < preciomenor
F
diamenor ← día
preciomenor ← precio
día ← día + 1
F
día ≥ diasmes
V
promedio ← total/diasmes
promedio
diamenor
Fin

```
Proceso promediopreciob

Leer diasmes
    total←0
    dia←0
    rango←(150-130)+1
    diamenor←0
    preciomenor←150
Repetir
    precio←TRUNC(AZAR(rango))+130
    total←total+precio
Si precio<preciomenor Entonces
    diamenor←dia
    preciomenor←precio
FinSi
    dia←dia+1
Hasta Que dia>=diasmes
    promedio←total/diasmes
Mostrar promedio
Mostrar diamenor
FinProceso
```

Otra forma de realizar el algoritmo consiste en cambiar la perspectiva del lazo repita-hasta por un mientras repita. Para el cambio será necesario solo negar la expresión usada en el lazo repita-hasta en días<diasmes.

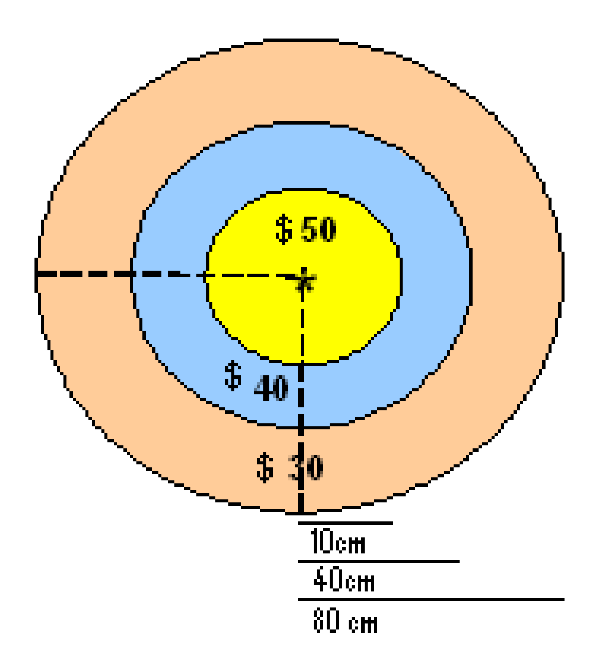


Proceso promediopreciоab

```
Leer diasmes
    total←0
    dia←0
    rango←(150-130)+1
    diamenor←0
    preciomenor←150
Mientras dia<diasmes Hacer
    precio←TRUNC(AZAR(rango))+130
    total←total+precio
Si precio<preciomenor Entonces
    diamenor←dia
    preciomenor←precio
FinSi
    dia←dia+1
Fin Mientras
    promedio←total/diasmes
Mostrar promedio
Mostrar diamenor
FinProceso
```

Ejercicio 2

"Tiro al blanco" es un juego que consiste en lanzar dardos a un objetivo circular. El premio que gana el jugador, depende de la ubicación en la cual cae el dardo y su valor se reparte en dólares ($30, $40 o $50), tal como se muestra en la figura:



Existen 3 círculos concéntricos (que tienen el mismo centro) y las longitudes de los radios del primero, segundo y tercer círculos son 10cm, 40cm y 80cm, respectivamente. Suponga que los 3 círculos están inscritos en un cuadrado de longitud de lado 160cm.

Escriba un algoritmo que permita simular **n** lanzamientos aleatorios de dardos, asignando

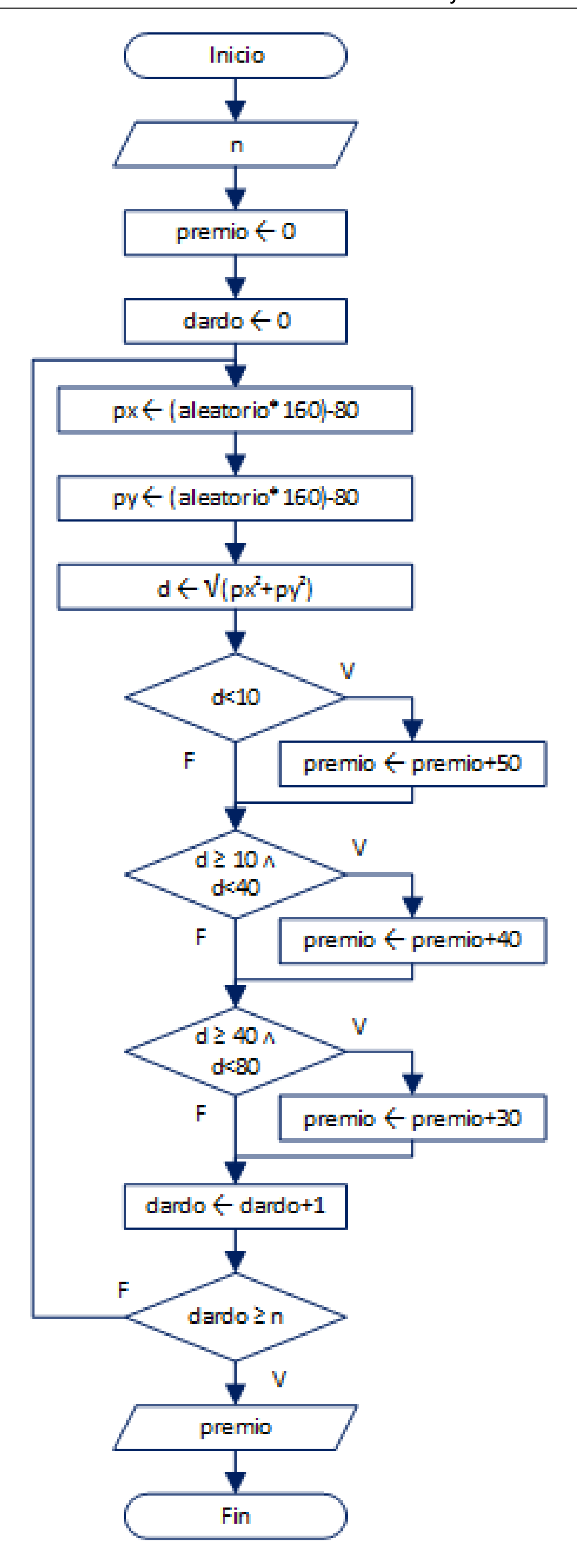
de forma aleatoria pares ordenados (x, y) en el cuadrado descrito. En cada lanzamiento se debe verificar si el dardo se ubica al interior de alguno de los círculos descritos y asignar el respectivo premio. Al final, muestre el premio total en dólares que obtuvo el jugador.

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación I Término 2007 - 2008. Julio 03, 2007

Desarrollo: Como bloque de ingreso, se usará la variable n como el número de dardos cuyo lanzamiento será simulado. Las coordenadas de los puntos donde cae cada dardo serán referenciadas al centro de los círculos, por lo que los rangos para el eje x y el eje y serán [-80, 80].
Si las coordenadas serán aleatorias el rango del aleatorio será 80-(-80)=160, con valor inicial de -80 como se describe en la siguiente expresión:

Coordenada px ← entero(aleatorio*160)-80

Al calcular la distancia de la coordenada del dardo al origen, usando la fórmula de distancia entre dos puntos, se determinará la franja sobre la que cayó el dardo.


Inicio
n
premio ← 0
dardo ← 0
px ← (aleatorio* 160)-80
py ← (aleatorio* 160)-80
d ← √(px²+py²)
d<10
V
F
premio ← premio+50
d ≥ 10 ∧ d<40
V
F
premio ← premio+40
d ≥ 40 ∧ d<80
V
F
premio ← premio+30
dardo ← dardo+1
dardo ≥ n
F
V
premio
Fin

En seudo-código el algoritmo se expresa como:

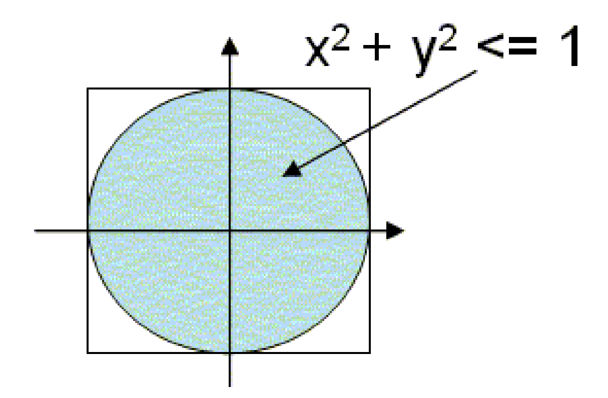
```
Proceso tabladardosa
Leer n
    premio←0
    dardo←0
Repetir
    px←(AZAR(160)-80)
    py←(AZAR(160)-80)
    d←RAIZ(px^2+py^2)
Si d<10 Entonces
    premio←premio+50
FinSi
    Si d>=10& d<40 Entonces
    premio←premio+40
FinSi
Si d>=40& d<80 Entonces
    premio←premio+30
FinSi
    dardo ← dardo+1
Hasta Que dardo>=n
Mostrar premio
FinProceso
```

Como ejercicio se propone realizar el diagrama de flujo usando el lazo Mientras-Repita:

```
Proceso tabladardosb
Leer n
    premio←0
    dardo←0
Mientras dardo<=n Hacer
    px←(AZAR(160)-80)
    py←(AZAR(160)-80)
    d←RAIZ(px^2+py^2)
Si d<10 Entonces
    premio←premio+50
FinSi
Si d>=10& d<40 Entonces
    premio←premio+40
FinSi
Si d>=40& d<80 Entonces
    premio←premio+30
FinSi
    dardo ← dardo+1
FinMientras
Mostrar premio
FinProceso
```

Ejercicio 3
Encuentre un valor aproximado de la constante π con el siguiente procedimiento:

Considere un círculo de radio unitario, centrado en el origen e inscrito en un cuadrado:



Para n puntos (**x**, **y**) con coordenadas generadas de forma aleatoria reales entre 0 y 1, determine cuántos puntos caen dentro del cuadrante de círculo.
Si llamamos a este contador **k**, se puede establecer la siguiente relación aproximada suponiendo **n** grande:

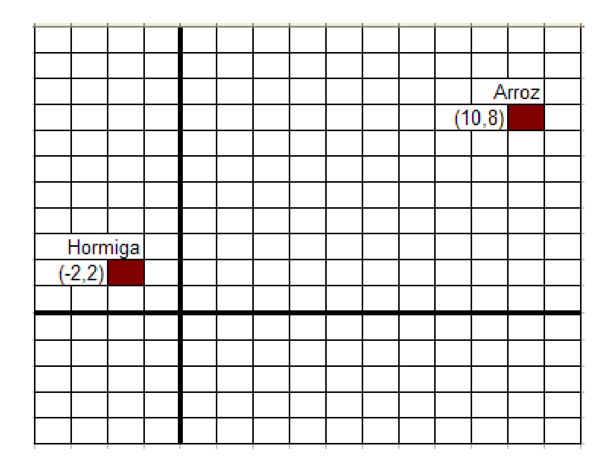
$$\frac{k}{n} = \frac{\left(\frac{1}{4}\, del\ área\ del\ círculo\right)}{\left(\frac{1}{4}\, del\ área\ del\ cuadradro\right)} = \frac{\left(\frac{1}{4}\right)\pi(1)^2}{\left(\frac{1}{4}\right)(2)^2} = \frac{\pi}{4}$$

Donde se puede obtener el valor aproximado de π

Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial II Término 2004 - 2005. Diciembre, 2004. Tema 3

Ejercicio 4

En un plano cartesiano se encuentran una hormiga y un grano de arroz. En cada instante de tiempo, la hormiga de manera aleatoria intuye la dirección donde ir (arriba, abajo, derecha, izquierda), y cuantas unidades desplazarse (entre 1 a 3) en la anterior dirección.
Implemente un algoritmo que simule 100 instantes de tiempo con desplazamientos de la hormiga que inicialmente se encuentra en las coordenadas (-2,2) y un grano de arroz en las coordenadas (10,8)
Al final indique las respuestas a las siguientes preguntas:
1. ¿La hormiga llegó al grano de arroz?
2. Si la respuesta a la pregunta anterior es "Si", entonces mostrar: cuántos pasos fueron necesarios.
3. ¿La distancia más lejana en la que estuvo la hormiga del grano de arroz?



Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. 1ra Evaluación II Término 2007 - 2008. Diciembre 04, 2007. Tema 1.

Ejercicio 5

Para una nueva versión del juego "Escaleras y Serpientes" se desea disponer del algoritmo para simulación en computador.

El juego de dos jugadores consiste en llegar a la meta en primer lugar en un tablero de 64 casillas cuyas especificaciones son las siguientes:

1. Cada jugador realiza su recorrido alternadamente de acuerdo a los resultados de los lanzamientos de un dado (6 caras)
2. Al avanzar, el jugador puede "caer" en una "casilla de castigo", por lo que retrocederá 3 pasos de la posición en la que se encuentra. Si cae en "casilla de premio", el usuario avanzará 3 pasos de la posición en la que se encuentra.
3. Luego de un lanzamiento y determinación de la posición final, el jugador le pasa el turno al otro jugador.
4. Se repite el juego desde el paso 2 hasta que uno de los jugadores pase la meta.

Al final se deberá mostrar:

- Número de veces jugadas por cada jugador, y
- El jugador que ganó.

Nota: casillas de premio para éste tema son: 4, 9, 29, 34, 46 y de castigo: 8, 19, 38, 50, 60



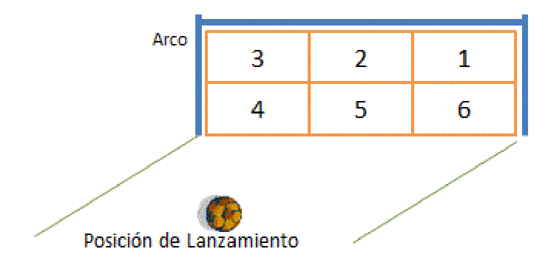
Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial II Término 2005 - 2006. Diciembre 06, 2005. Tema 4

Ejercicio 6
En el Fútbol el lanzamiento de penales intervienen el jugador que patea y el arquero que tapa el penal.
Este juego consiste en 5 lanzamientos por parte de los jugadores que patean el balón, los cuales pueden decidir lanzar en cualquiera de las seis secciones del arco (1: arriba a la derecha, 2: arriba al centro, 3: arriba a la izquierda, 4 abajo a la izquierda, 5: abajo al centro, 6: abajo a la derecha). En cada lanzamiento, el arquero decide donde ubicarse para atajar el tiro y no tiene oportunidad de cubrir otra sección, si éste coincide con la ubicación donde disparó el jugador, entonces el lanzamiento fue atajado o fallado, caso contrario se marcó un GOL.

Escriba un algoritmo que simule un juego de 5 lanzamientos de penales, en donde la sección del arco donde cada jugador lanza es decidido por el usuario y la sección cubierta por el arquero es simulado por el computador (aleatoria).

Al final presente la siguiente información:

- Cantidad de goles conseguidos.
- Cantidad de penales fallados.
- La cantidad de goles realizados en la parte derecha, central e izquierda del arco.
- La ubicación del arco (derecha, centro o izquierda) por donde ingresaron más goles. Suponga que existe una sola.
- La ubicación del arco (derecha, centro o izquierda) por donde no ingresaron goles. Suponga que existe una sola.



Referencia: ESPOL-FCNM. ICM00794-Fundamentos de Computación. Parcial I Término 2005 - 2006. Julio 05, 2005. Tema 4

5 — Subalgoritmos

Laura M. Angelone

Muchas veces es más elegante y barato mejorar un algoritmo que comprar una computadora más rápida. Los dilemas algorítmicos están presentes en todas las actividades humanas y resolverlos en forma rápida y eficiente puede ser un desafío entretenido si gustamos de la lógica y la matemática.

R. Baeza-Yates

5.0.3 Introducción

A continuación desarrollaremos el concepto de subalgoritmos. Estos módulos son esenciales a la hora de tratar la resolución de problemas complejos, donde se requiere la división del problema en sub-problemas de menor complejidad. Paralelamente, la utilización de subalgoritmos requiere la definición de conceptos tales como variables locales y globales, parámetros y argumentos, a fin de poder relacionar los subalgoritmos entre sí y con el algoritmo principal.

5.1 Programación modular

La experiencia ha demostrado que un problema complejo se resuelve mejor si se lo divide en sub-problemasde menor complejidad, es decir, dado unproblema se debe pensar en sub-problemas que constituyan módulos separados que luego se ensamblen para construir el algoritmo que resuelva el problema de origen.Las características de los módulos deben ser:

- cada módulo debe ser independiente de los otros,
- cada módulo debe estar definido en sí mismo, con sus entradas, sus acciones y sus resultados.

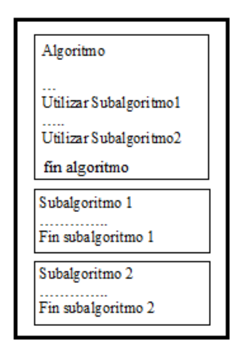
Partiendo de estas características, la ventaja de la modularidad es que se pueden probar los diferentes módulos en forma independiente. Se los puede modificar sin afectar a los otros. Y, ademas, permite que diferentes diseñadores puedan trabajar simultáneamente en cada uno de ellos.vez escritos y probados, los módulos son ensamblados adecuadamente para lograr la resolución del problema inicial.Por otro lado, el concepto de modularización se puede asociar al concepto de reuso. En la resolución de problemas complejos se pueden utilizar partes de otros problemas previamente resueltos, esto es, reutilizar soluciones ya conocidas y testeadas con anterioridad.Un algoritmo o parte de él que puede ser reutilizado en otros contextos de problemas algorítmicos recibe el nombre de **módulo** o **rutina**. Una rutina, según el diccionario de la Real Academia Española, es una secuencia invariable de instrucciones que forma parte de un programa y se

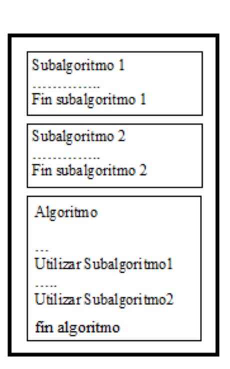
puede utilizar repetidamente.Los programadores expertos emplean buena parte de su tiempo en identificar rutinas y reutilizarlas en forma apropiada.Cada **módulo** es parte de un algoritmo general, por lo cual lo denominaremos **subalgoritmo**.

5.2 Concepto de subalgoritmo

*Un **subalgoritmo** define una tarea específica que puede ser reutilizada en distintos contextos de resolución de problemas algorítmicos*

Algunos módulos son tan comunes que están incluidos en los lenguajes de programación. Son las denominadas funciones internas, tales como **sin**( ), **cos**( ), **abs**( ), las cuales son suministradas en forma de biblioteca o paquetes de módulos reutilizables.
Como se mencionó, cada subalgoritmo puede desarrollarse en forma independiente de los restantes. Debe tener una zona de ingreso de datos y otra de salida de resultados.
Los esquemas que se muestran a continuación representan dos posibles estructuras equivalentes de diseño de algoritmos utilizando subalgoritmos. Notar que es lo mismo pensar y escribir primero los subalgoritmos, y luego utilizarlos en el algoritmo principal, o primero pensar y escribir el algoritmo principal proponiendo los subalgoritmos, y luego escribir cada uno de ellos, como se muestra en Fig.1. Es decir, es indistinto el orden en que se piensan y escriben el algoritmo y subalgoritmos que componen el mismo.





1. Es indistinto el orden enque se piensan y escriben el algoritmo y subalgoritmo
Se diferencian dos tipos de subalgoritmo acorde a las tareas que desarrollan: los subalgoritmos **función** y los subalgoritmos **subrutina**, los que veremos a continuación.

5.3 Subalgoritmo función

Este tipo de subalgoritmos se implementa cuando la tarea a desarrollar devuelve un único resultado, en general son cálculos. Responden al concepto matemático de función de una o más variables. Ejemplos de este tipo de subalgoritmos son las funciones internas tales comosin( ), cos( ), abs( ), hallar el máximo de un conjunto de números, etc.

Un subalgoritmo función es un subalgoritmo que recibiendo o no datos devuelve ***un único resultado****.*

Una función es un conjunto de acciones (una o varias) con un **nombre**, un **tipo de dato** y un **resultado único**.

5.3.1 Sintaxis de la declaración de funciones

función *<nombre de la función>(<lista de parámetros>): <Tipo de resultado>*
<declaración de variables>
Inicio
Cuerpo <acciones>
Devolver (<constante, variable o expresión>)
fin función

Nombre de función: el nombre de una función sigue la misma regla de los nombres de variables, es decir, debe ser un identificador válido.
Tipo de resultado: señala el tipo de dato de la información que devuelve la función.
Lista de parámetros: contiene las variables que pasan información necesaria para que la función ejecute el conjunto de acciones. Cada variable de la lista debe tener una declaración de tipo dentro del paréntesis. Puede que no hayalista de parámetros, en dicho caso se escribe el paréntesis vacío.
Declaración de variables: en este lugar se deben declarar aquellas variables que se usarán en la función, salvo las que están declaradas en la lista de parámetros. Estas variables son llamadas **locales** dado que su alcance se restringe a la función. Cuando se abandona la función dichas variables no pueden ser accedidas.
Cuerpo de la función: lo constituye el conjuntode acciones a realizar por la función.
Retornar el resultado: el único resultado que devuelve la función puede ser un valor constante, o una variable o una expresión válida, la cual debe colocarse entre paréntesis al lado de la acción Devolver. Cuando se ejecuta esta acción se *devuelve* el control del programa al lugar donde se ha llamado a la función.
Ejemplos de subalgoritmos **función**
A continuación veremos algunos ejemplos de funciones y su utilización.

Ejemplo 1: Definir la función***y =x*** donde el exponente *n* es de tipo entero
El pseudocódigo de resolución de esta función es el siguiente:

```
función potencia ( x:real , n:entero ):real
variables
    entero i
    real y
inicio
    y ←1
    Repetir para i←1 hasta abs(n) hacer
        y ← y * x
    finpara
    si n < 0 entonces
```

```
        y ← 1 / y
    fin si
     Devolver (y)
fin función
```

5.3.2 ¿Cómo usar/llamar/invocar una función?

A este tipo de subalgoritmo se lo puede llamar o invocar desde una expresión, desde una asignación, desde una decisión, o una salida de datos. Debe cumplirse que el valor calculado que devuelve la función en su nombre sea recibido en un contenedor válido respecto de su tipo de dato.

Ejemplo 2: Escribir un algoritmo que utilice la función del ejemplo anterior para calcular las 10 primeras potencias de un valor x de tipo real que primero deberá ser ingresado por teclado.A continuación se muestran 2 modos diferentes de llamar a la **función potencia**.

En el Modo 1 la función potencia es empleada directamente dentro de la acción escribir, mientras que en el Modo 2 el valor devuelto por la función potencia es asignado en la variable real z y es el valor almacenado en z el que luego se muestra.

```
Modo 1
Algoritmo calcul_potencia
var     entero i
        real x
inicio
        Escribir ('Evalúa la función  y = x ^ n ')
        Escribir ('Ingrese valor de x')
        Leer (x)
        Repetir Para  i ←1 hasta 10 hacer
            Escribir ("x ^", i, " : ", potencia (x , i ) )
        finpara
Fin
```

```
Modo 2
Algoritmo calcul_potencia
var     entero i
        real x
inicio
        Escribir ('Evalúa la función  y = x ^ n ')
        Escribir ('Ingrese valor de x')
        Leer (x)
        Repetir Para  i ←1 hasta 10 hacer
            z ← potencia (x, i)
            Escribir ("x ^", i, " : ", z )
        finpara
Fin
```

Las diferentes formas de invocar/llamar a un subalgoritmo función son las siguientes:

Asignando el resultado de la función a una variable, como es el Modo 2.
Dentro de una acción Escribir, como es el Modo 1.
Como parte de una expresión, por ej. Z← 2*potencia(w, 7)/ 10.
Dentro de una condición, por ej. Repetir mientras potencia(w,n) < 100 hacer .

Veamos otros ejemplos:

Ejemplo 3: Desarrollar un subalgoritmo que calcule el factorial de un número entero **n**.
El pseudocódigo de resolución de esta función es el siguiente:

```
función FACTORIAL ( n:entero):real
variables
    entero i, f
inicio
    f ← 1
    Repetir para i ← 1 hasta abs(n) hacer
```

f ← f * i
finpara
Devolver (f)
fin función

Ejemplo 4: Realizar un subalgoritmo que verifique si una fecha es válida o no.Esta función de tipo lógica o booleana, necesita recibir 3 parámetros (día, mes y año) y devuelve verdadero o falso, según si la fecha recibida es válida o no.Debe considerarse que el año puede ser bisiesto. El pseudocódigo de resolución de esta función es el siguiente:

función Validación-de-fecha (dia, mes, anio:entero): **boolean**
variables
boolean correcta, bisiesto
inicio
bisiesto ← falso
Si (anio MOD 4= 0) AND (anio MOD 100 <> 0) OR (anio MOD 400= 0)
entonces bisiesto ← verdadero
finsi
correcta ← verdadero
Segúnsea mes hacer
1,3,5,6,8,10,12: Si dd>31 entonces correcta ← falso
finsi
4,7,9,11: Si dia > 30 entonces correcta ← falso
finsi
2: Si bisiesto entonces Si dia > 29 entonces correcta ← f
finsi
sino
Si dia>28 entonces correcta ← falso
finsi
físi
valores: correcta ← falso
finsegún
Devolver (correcta)
fin de la función Validación-de-fecha

5.4 Subalgoritmo subrutina

El empleo de los subalgoritmos función está limitado a los casos donde el resultado debe ser expresado por un único valor. Sin embargo, en ciertas ocasiones se necesita que el subalgoritmo devuelva más de un valor (por ejemplo raíces de un polinomio) o no devuelva nada (por ejemplo una impresión), en este caso se introduce una nueva idea de subalgoritmo que se denomina textbfsubrutina.

En general se piensa en una subrutina cuando la tarea a desarrollar es un proceso que no necesariamente tenga un único resultado -aunque no es excluyente- tales como impresión, ordenamiento, búsqueda de información, dibujo de figuras o interfaces, entre otros.

Una subrutina es un subalgoritmo que recibiendo o no datos permite devolver varios resultados, un resultado o ninguno

5.4.1 Sintaxis de la declaración de subrutinas

La sintaxis es la siguiente:

subrutina *<nombre de la subrutina>(**E**:<lista de parámetros de entrada>, **S**: <lista de parámetros de salida>)*
variables <declaración de variables locales>
inicio
cuerpo *<acciones>*
fin subrutina

Nombre de la subrutina: el nombre de una subrutina sigue las mismas reglas de los nombres de las variables.
Lista de parámetros: contiene las variables que pasan alguna información que necesita la subrutina para ejecutar sus acciones y aquellas variables en las cuales se almacenarán los resultados. Cada variable de la lista debe tener una declaración de tipo dentro del paréntesis. Además, se debe escribir la letra E y dos puntos antes de la lista de parámetros de entrada (datos), y *S* y dos punto*s* antes de la lista de los parámetros de salida (resultados). En algún caso puede que no haya lista de parámetros en cuyo caso se escribe el paréntesis vacío.
Declaración de variables locales: se deben declarar aquellas variables que se usarán en la subrutina, salvo las que están declaradas en la lista de parámetros.
Cuerpo de la función: lo constituye el conjunto de acciones a realizar por la subrutina.
Retornar el resultado: la subrutina no retorna en una acción *Devolver* como la función, pues puede devolver más de un valor, o sólo uno o ninguno, y lo hace a través de sus parámetros de salida.A continuación veremos algunos ejemplos de subalgoritmos subrutina.

Ejemplo 1: Realizar la función ***x*** como una subrutina
El pseudocódigo de resolución de esta subrutina es el siguiente:

```
subrutina XalaN ( E: x:real, n:entero; S: y: real)
variables
    entero i
inicio
    y ← 1
    Repetir para i←1 hasta abs (n) hacer
        y ← y * x
    finpara
    Si n < 0 entonces y ← 1 / y
    finsi
fin subrutina
```

5.4.2 ¿Cómo usar/llamar/invocar una subrutina?

Cuando un algoritmo llama a una subrutina lo hace escribiendo el nombre de la misma seguido de los argumentos (datos y variables resultados) como si se tratase de una acción más.

nombre de la subrutina(lista de argumentos de entrada, lista de variables de resultado)

A una subrutina se la llama desde una línea del algoritmo, desde una sentencia en un programa.
Al aparecer el nombre de la subrutina se establece la correspondencia entre los argumentos (en la llamada a la subrutina) y los parámetros (subrutina propiamente dicha) y se ejecutan las acciones indicadas en el cuerpo de la subrutina. Cuando finaliza la ejecución de la subrutina, se devuelve el control del proceso al algoritmo que realizó la llamada. Y éste continúa en la acción inmediatamente siguiente a aquella en la cual figura el nombre de la subrutina. A continuación veremos algunos casos que ejemplifican este mecanismo.

Ejemplo 2: Escribir un algoritmo que utilice la función del ejemplo anterior para calcular las 10 primeras potencias de x. El valor de x deberá ser ingresado por teclado, siendo x un valor real.

```
Algortimo calcul_potencia
variables
    entero n
    real x, r
inicio
    Escribir (‘Tabla y = x ^n ’)
    Escribir (‘Ingrese valor de x’)
    Leer (x)
    Repetir para i←1 hasta 10 hacer
        XalaN ( x, i, r )
        Escribir (“x ^”, i,“:”, r )
    finpara
fin
```

Notar que existe una única forma de invocar a las subrutinas, “escribiendo el nombre de la subrutina seguido de los argumentos entre paréntesis”, como si fuera una acción más - tales como el Leer(..) o el Escribir(...).

5.4.3 Subrutinas que no devuelven ningún resultado

Para desarrollar este tema veremos algunos ejemplos que mostrarán su uso.

Ejemplo 3: Escribir un subalgoritmo subrutina que muestre un encabezado

```
subrutina Encabezado( )
inicio
    Escribir (‘******************************’)
    Escribir (‘ INFORMATICA I ’)
    Escribir (‘******************************’)
fin subrutina
```

Ejemplo 4: Escribir un subalgoritmo subrutina que muestre una línea de 25 asteriscos

```
subrutina TrazalineaI ( )
variables
    entero i
inicio
    Repetir para i ← 1 hasta 25 hacer
        Escribir (‘*’)
    finpara
fin subrutina
```

Entonces la subrutina Encabezado( ) se podría escribir utilizando TrazalineaI, resultando:

```
subrutina Encabezado ( )
inicio
    TrazalineaI ( )
    Escribir (‘ INFORMATICA I ’)
    TrazalineaI ( )
fin subrutina
```

Ejemplo 5: Escribir un subalgoritmo subrutina que muestre una línea de asteriscos de longitud a requerimiento del usuario.

```
textbfsubrutina TrazalineaII ( E: long: entero )
variables
    entero i
inicio
    Repetir para i←1 hasta long hacer
        Escribir (‘*’)
    finpara
fin subrutina
```

Entonces la Subrutina Encabezado( ) se podría escribir utilizando TrazalineaII, resultando:

```
subrutina Encabezado ( )
inicio
    TrazalineaII ( 25 )
    Escribir (‘ INFORMATICA I ’)
    TrazalineaII ( 25 )
fin subrutina
```

Ejemplo 6: Escribir un subalgoritmo subrutina que muestre una línea de longitud y carácter a requerimiento del usuario.

```
subrutina TrazalineaIII (E: long: entero, simb: char)
variables
    entero i
inicio
    Repetir para i ←1 hasta long hacer
        Escribir ( simb )
    finpara
```

fin subrutina

Entonces el Subrutina Encabezado( ) se podría escribir utilizando TrazalineaIII, resultando:

subrutina encabezado ( )
inicio
    TrazalineaIII (25, '*')
    Escribir (' INFORMATICA I ')
    TrazalineaIII (25, '*')
fin subrutina

5.4.4 Un caso especial: los parámetros de entrada también son de salida

Existen casos donde los parámetros de entrada también sirven como parámetros de devolución de resultados.Veremos a continuación un ejemplo y su utilización.
Ejemplo 7: Desarrollar una subrutina que permita realizar el intercambio de los valores de 2 variables numéricas.

subrutina intercambio (E/S: a, b: real)
variables
    real aux
inicio
    aux ← a
    a ← b
    b ← aux
fin subrutina

A continuación veremos un ejemplo de la utilización de esta subrutina.

Ejemplo 8: Desarrollar un algoritmo para ordenar tres números reales utilizando la subrutina intercambio.

Algoritmo ordenar
variables
    real x, y, z
inicio
    Escribir ('Ingrese tres números reales ')
    Leer (x, y, z)
    Si x > y entonces intercambio (x, y)
    finsi
    Si y > z entonces intercambio (y, z)
    finsi
    Si x > y entonces intercambio (x, y)
    finsi
    Escribir (x, y, z)
Fin.

5.5 Parámetros y argumentos en los subalgoritmos

Las variables que aparecen en el enunciado de un subalgoritmo, conforman la lista de **parámetros**

Los parámetros sólo deben ser variables, pues reciben los datos que se envían en la llamada y variables donde se depositan los resultados.

Las variables correspondientes a la llamada al subalgoritmo se denominan **argumentos**

Los argumentos contienen los valores necesarios para evaluar el subalgoritmo en ese momento (correspondiente a los datos de entrada) y en las subrutina se agregan las variables que recibirán los valores de los resultados de la subrutina (correpondiente a las salidas).
Los argumentos de entrada (***datos***) tanto en funciones como en subrutinas pueden ser constantes, variables, expresiones, valores de funciones y hasta nombres de funciones o subrutinas.

Los argumentos que recibirán los resultados (***salida***) en las subrutinas deben ser nombres de variables.

Cada vez que un subalgoritmo es llamado desde el algoritmo o desde otro subalgoritmo, se establece una **correspondencia** entre los argumentos y los parámetros de tipo posicional, es decir, el primer argumento se corresponde con el primer parámetro, el segundo con el segundo y así sucesivamente. Debe darse también la consistencia de tipos entre el argumento y su correspondiente parámetro. Si esto no se cumple, el compilador detectará el error. Y la última correspondencia es de cantidad, si el subalgoritmo tiene *n* parámetros implica que al llamarlo debe tener *n* argumentos.

Ejemplo 1: Parámetros y argumentos en la función *Cuenta*

función Cuenta(x1, x2, t: real) : real
variables
    real z
inicio
    x1 ← x1 *100
    z ← (x1 + x2) * t
    devolver (z)
fin función

La utilización o llamada a la función podría ser de la siguiente manera:
temp ← x * y * tiempo + **Cuenta**(x, y, tiempo)
En este caso la correspondencia entre argumentos y parámetros es x con x1, y con x2, tiempo con t

Otra llamada podría ser:
temp ← x * y * tiempo + **Cuenta**(1.5*x, (x+z)*2.5, tiempo)

Como conclusión podemos decir que los **parámetros** *van con las definiciones de los subalgoritmos ylos* **argumentos** *con las llamadas a los mismos*

Ejemplo 2: Parámetros y argumentos en la subrutina *división*

subrutina división (E: dividendo, divisor: entero; S: cociente, resto: entero)
inicio
cociente ← dividendo div divisor
resto ← dividendo mod divisor
fin subrutina división

La utilización o llamada a la subrutina podría ser de la siguiente manera:
división (m,n,p,q)
En este caso la correspondencia entre argumentos y parámetros es m con dividendo, n con divisor, p con cociente, q con resto.

Otra llamada podría ser:
división (m*n-4, n+1, s, t)

En este caso la correspondencia entre argumentos y parámetros es m*n-4 con dividendo, n+1 con divisor, s con cociente, t con resto. Se debe notar que los 2 últimos argumentos *deben* ser variables pues reciben información desde la subrutina.

5.6 Memoria para los subalgoritmos

Siempre que un subalgoritmo es llamado se crea un espacio de memoria principal (RAM) temporal para cada uno de los parámetros y variables locales del mismo. Cada parámetro es inicializado con el valor del argumento correspondiente en el enunciado de llamada. Después de esta inicialización, el procesador ejecuta el cuerpo del subalgoritmo.
Las celdas de memoria del subalgoritmo pueden ser pensadas como espacios temporales de escritura, porque cuando el procesador finaliza el subalgoritmo, estas celdas de memoria son liberadas y sus contenidos se pierden.Aclaración: la palabra temporal aquí tiene significado de “poco tiempo”, momentáneo.

5.7 Parámetros formales y actuales con igual nombre

Ya sabemos que las variables que usa un subalgoritmo residen en memoria principal de tipo RAM, y con ellas trabaja el procesador en su ejecución.

Cuando se llama a un subalgoritmo, el procesador le dispone de un sector de memoria distinto al del programa llamador. Por ello el subalgoritmo tiene su propio segmento de memoria para colocar los parámetros formales y toda otra variable que necesite.

¿Qué pasa cuando en el algoritmo existen variables cuyos nombres coinciden con algunas del subalgoritmo? La respuesta es simple: **no hay problema**. Esto es debido a que están en lugares de memoria distintos, en distintos ámbitos. Por esta misma razón los nombres de los parámetros y los argumentos pueden ser distintos o no.

Veamos un ejemplo con la función ***Potencia*** ya definida.

```
Algoritmo calcular_potencias
    variables
        entero i
        real x

inicio
    Escribir (“Evalúa función y=xcon n de 1 a 10”)
    Escribir (“Ingrese el valor de x : ”)
    Leer (x)
    Repetir Para i <– 1 hasta 10 hacer
        Escribir(x, “",i,“=",Potencia(x,i))   finparaFin.función Potencia(x,n):realvariables   enterox,n,i   realyinicio   y ←
1
    Repetir para i ← 1 hasta abs (n) hacer
        y ← y * x
    finpara
    Si n < 0 entonces
        y ← 1 / y finsi
    Devolver (y)
fin función
```

5.8 Variables locales y globales

***Variables globales o externas**: son aquellas que se declaran en el algoritmo que llama al subalgoritmo, y su uso no sólo abarca al algoritmo sino que también pueden ser utilizadas en el subalgoritmo.*

En este caso las subrutinas y funciones "ven"las variables globales y pueden usarlas sin necesidad de declararlas, las usan directamente desde la memoria principal (RAM)del algoritmo.

***Variables locales o internas**: Las variables cuya validez alcanza sólo al subalgoritmo son locales a él.*

Es decir, son aquellas variables que se declaran en el subalgoritmo. Su uso se limita al subalgoritmo, y una vez que éste retorna al algoritmo que lo llamó, dichas variables quedan indefinidas y desaparecen. Los parámetros formales también son variables locales. Si se vuelve a llamar al subalgoritmo, las variables locales no tienen porque conservar los valores que tuvieran antes.Veamos un ejemplo:

```
Algoritmo A                 Función B( h : real ): real
variables real x, y , z     variables real r, x
inicio                      inicio
    x ← 2                       z ← z + h
    z ← 10                      x ← 7
    y ← B ( x )                 r ← z + x
    Escribir ( x, y, z )        Devolver ( r )
Fin.                        fin función
```

5.8.1 ¿Cómo funciona este algoritmo?

El algoritmo tiene 3 variables x, y, z . Se le asignan valores a: x y z , pero el valor de y se obtiene al llamar a la Función B . Notemos que la variable z aparece en la Función pero no está declarada en ella.Realicemos la prueba de escritorio:

La variables z es modificada por la función ya que la misma es una variables global.
Por lo tanto se imprime ***2 19 12***

Notamos que :

- los nombres de los parámetros y los argumentos no tienen porque ser iguales.
- x, y , z son variables globales; h, r , x son variables locales; z es una variable global que es usada por la función; existen dos variables x distintas: una está definida en la memoria que tiene asignado el algoritmo y la otra (local a la función) está definida en la memoria que tiene asignado el subalgoritmo función. Por lo tanto, al estar en lugares distintos, no son iguales. Si se desea que un conjunto de variables tenga "visibilidad .en todo el programa (algoritmo principal y subalgoritmos) las variables deben definirse como globales. En este caso se trabaja sobre una única posición de memoria y todos los módulos miran a esa posición al referenciar el nombre de la variable global. Este tipo de proceso debe usarse en casos muy específicos, pues el abuso genera programas difíciles de mantener, o resultados no deseados.

Realicemos la resolución de un sistema de 2 ecuaciones con 2 incógnitas.

Ejemplo 1: Realizar un algoritmo que reciba como datos los coeficientes a1,a2,b1,b2,c1 y c2 de un sistema de 2x2 de ecuaciones, llame a un subalgoritmo RESOLVER para resolverla y luego imprima el sistema con sus soluciones correspondientes.

5.8.2 Análisis del problema:

Datos: a1, a2, b1, b2, c1, c2 coeficientes de un sistema de ecuaciones
Resultados: x , y solución del sistema
Proceso de resolución: se propone utilizar el método de Cramer:

D= Discriminante = a1 * b2 - a2 * b1 Si D es distinto de 0 se obtienen soluciones únicas
DX = Discriminante X = c1 * b2 - c2 * b1DY = Discriminante Y = a1 * c2 - a2 * c1
La solución del sistema se determina de la siguiente manera:
x = DX / D
y = DY / D

Algoritmo Resolución de sistema de 2x2
variables real a1, a2, b1, b2, c1, c2 , x , y
inicio
Leer (a1, a2, b1, b2, c1, c2)
RESOLVER (a1, a2, b1, b2, c1, c2, x, y)
Escribir (“El sistema ”, a1, “x + ”,b1, “ y = ”, c1)
Escribir (“ ” , a2 ,“x + ”,b2, “ y = ”, c2)
Imprimir (“tiene solución x =”, x , “ y =”, y)
Fin.

Subrutina RESOLVER (E:a1,a2,b1,b2,c1,c2:real, S:sol1,sol2:real)

variables real D, DX, DY
inicio
D ← a1 * b2 - a2 * b1
DX ← c1 * b2 - c2 * b1
DY ← a1 * c2 - a2 * c1
sol1 ← DX / D
sol2 ← DY / D
fin subrutina

Notemos que :
a1, a2, b1, b2, c1, c2 , x , y son variables globales, pero ninguna es usada como tal en la subrutina, ya que ésta tiene variables a1, a2, b1, b2, c1, c2, sol1, sol2, D , DX y DY que son variables locales a ella.
a1, a2, b1, b2, c1, c2 del algoritmo no son las mismas que las de la subrutina.
sol1 y sol2 son variables de salida de resultados, y dichos resultados son tomados por x e y respectivamente del algoritmo.

Otra forma (no conveniente) de resolución de la subrutina sería usando a : a1, a2, b1, b2, c1, c2 como variables globales , a saber :

Subrutina RESOLVER (S:sol1,sol2: real)
variables real D, DX, DY
Inicio
D ← a1 * b2 - a2 * b1
DX ← c1 * b2 - c2 * b1
DY ← a1 * c2 - a2 * c1
sol1 ← DX / D
sol2 ← DY / D
fin subrutina

5.9 Pasaje de información entre argumento y parámetro

Como ya mencionamos, cada vez que un algoritmo llama a un subalgoritmo, se establece una correspondencia automática entre los argumentos y los parámetros correspondientes.

En el pasaje de estos parámetros se distinguen básicamente dos formas: pasaje por valor y pasaje por referencia.

Pasaje de parámetros por valor: es la forma más simple de pasar los datos, donde los parámetros reciben como valores iniciales una copia de los valores de los argumentos y con estos valores se comienza a ejecutar el subalgoritmo.Por ejemplo a1, a2, b1, b2, c1, c2 del ejemplo dado en el subcapítulo anterior: Algoritmo de resolución de un sistema de 2x2 de ecuaciones. Es decir, el valor del argumento es pasado al parámetro correspondiente en su área de memoria, en definitiva es una acción de asignación. Las funciones reciben los datos a través de este tipo de pasaje. También lo hacen los parámetros de entrada en una subrutina.

Pasaje de parámetros por referencia: en lugar de pasar el valor del argumento como valor inicial para su respectivo parámetro, el pasaje de parámetros por referencia, también llamado por dirección, establece una conexión directa a través de su dirección de memoria. Para distinguir

este tipo de pasaje nosotros usaremos una letra **"S"** en la lista de parámetros.
Las variables de salida en una subrutina tienen este tipo de pasajes. También los arreglos son pasados por referencia para ahorro de memoria.
Un ejemplo de pasaje por referencia son las variables sol1 y sol2 con x e y de dicho Algoritmo.

5.10 Menú

Hasta ahora realizamos los algoritmos por separado sin ningún nexo de unión entre ellos. Generalmente esto no sucede así, en problemas más complejos aparecen relaciones entre distintos algoritmos, y se debe escribir uno que indique cuál es el que deseamos ejecutar en ese instante: el algoritmo **principal**. Esto es posible gracias a la existencia de los subalgoritmos.

Un **menú** *consiste en presentar en pantalla una serie de opcionespara realizar acciones determinadas*

En cada opción podremos colocar la llamada a un subalgoritmo que cuando se termine de ejecutar, el algoritmo vuelve de nuevo a presentar el menú del que había partido.
A veces los menús se presentan anidados, es decir, alguna de las opciones del menú al ser seleccionada, hace que aparezca otro menú, dando lugar a nuevas posibilidades de elección.

Los menús permiten ejecutar más de un programa, sin necesidad de tener que escribir su nombre, cada vez que se desea ejecutarlo. Simplemente, le indicaremos mediante una variable la opción deseada.

La selección de opciones del algoritmo generalmente se realiza con la estructura de selección múltiple Según sea.

La idea es la que se observa en el siguiente cuadro:

```
Repetir
        Escribir ("Menú de opciones")
        Escribir ("1: Ingresar un nuevo artículo ")
        Escribir ("2: Buscar precio por artículo ")
        Escribir ("3: Modificar precios ")
        Escribir ("4: Dar de baja un artículo")
        Escribir ("0: FIN ")
        Escribir ("Ingrese su opción")
        Leer ( opción )
        Según sea opción hacer
                1: INGRESO
                2: BUSQUEDA
                3: MODIFICAR
                4: SACAR
        finsegún
hasta que opción = 0
```

Donde INGRESO, BUSQUEDA, MODIFICAR Y SACAR son llamadas a subrutinas.

De esta manera el algoritmo principal queda entendible y se saca del mismo todo proceso extra. Esta es una forma de modularizar un problema

A continuación desarrollaremos el siguiente problema:

“Realizar un algoritmo que permita convertir grados, minutos y segundos a segundos y vice-versa”

```
Algoritmo GMS a Seg y viceversa
variables
    entero opcion, G,M,S,seg
Inicio
    Repetir
    Escribir (‘Menu: conversión de ángulos’)
    Escribir (‘1-Convertir G-M-S a seg’)
    Escribir (‘2- Convertir seg a G-M-S’)
    Escribir (‘3- Salir’)
    Leer (opcion)
    VALIDAR (1,3,opcion)
    Según sea opcion hacer
        1: Escribir (‘Ingrese un ángulo en G-M-S’)
        Leer (G)
        VALIDAR (0,360,G)
        Leer (M)
        VALIDAR (0,59,M)
        Leer (S)
        VALIDAR (0,59,S)
        Escribir (‘el ángulo ingresado es: ’, GMSaS(G,M,S), ‘segundos     ’)
        2: Escribir (‘Ingrese un ángulo en seg’)
        Leer (seg)
        SaGMS (seg,G,M,S)
        Escribir (‘el ángulo ingresado es: ’, G, ‘grados’, M, ‘minutos’,S, ‘segundos’ )
        3: Escribir (‘fin del programa’)
    Fin según
    hasta que opcion = 3
Fin.

Subrutina VALIDAR (E: vi, vf: entero; S: Z: entero)
Inicio
    Repetir mientras (Z < vi) O (Z > vf) hacer
        Escribir (‘Error. Ingrese de nuevo el valor’)
        Leer (Z)
    Fin mientras
Fin subrutina

Funcion GMSaS ( entero GG,MM,SS) : entero
Inicio
    GMSaS = GG*3600 + MM * 60 + SS
```

Fin funcion

Subrutina SaGMS (**E:** segundos: entero; **S**: GG, MM,SS: entero)
Inicio
SS ←segundos mod 60 // o GG ←segundos div 3600
GG ←segundos div 60 div 60 // o MM ←segundos mod 3600 div 60
MM ←segundos div 60 mod 60 // o SS ←segundos mod 3600 mod 60
Fin subrutina

6 — Estructuras de datos

Gracia María Gagliano

La Máquina Analítica no tiene la pretensión de crear nada. Puede realizar cualquier cosa siempre que conozcamos cómo llevarla a cabo. Puede seguir análisis, pero es incapaz de descubrir relaciones analíticas o verdaderas. Su potencialidad es la de ayudarnos a hacer posible aquello sobre lo que tenemos un conocimiento previo.

Ada Byron (1815-1852)

6.0.1 Introducción

Desde los comienzos de este libro hemos visto que, para que la computadora pueda procesar un valor, ya sea numérico, carácter, sonido o imagen, éste debía estar almacenado en su memoria principal, expresado en cadenas de bits, cuya combinación se ajustaría a un mecanismo específico según el caso (no se representa igual un número entero que un real o un carácter o un sonido o una imagen).
Entonces, si para determinado proceso, fuera necesario mantener almacenado el valor que pudiera asumir un dato del mundo real del problema, se usaría un único lugar de memoria para representarlo. Si los posibles valores a procesar de este dato fueran varios, su procesamiento se haría en forma independiente, de a uno por vez; tal que cada nuevo valor sustituya al anterior; es decir, todos irían ocupando sucesivamente, el mismo lugar de memoria.Los datos así caracterizados se definieron como datos simples o tipos primitivos de datos.Muchos problemas describen situaciones en las que la unidad de información útil no aparece aislada en forma de dato simple, sino que lo hace como un grupo de datos que se caracterizan por la existencia de cierta vinculación entre ellos, tal que resulte adecuado pensarlos como un todo.Este enfoque nos lleva al concepto de dato estructurado o compuesto, como:

Un dato estructurado es un conjunto de datos agrupados bajo un mismo nombre y lógicamente vinculados de manera tal, que representen un comportamiento específico

Para poder procesar a los valores que lo componen deberá disponerse de un grupo de lugares de memoria que puedan ser ocupados simultáneamente. Sería apropiado imaginarlos como áreas contiguas, dispuestas de una forma determinada, que funcionaría como contenedor de los valores.Por ejemplo: Un punto en el espacio es un dato estructurado o compuesto pues para definirlo se requiere de una terna de números. Por lo tanto, para implementarlo en la memoria de una computadora, puede hacérselo mediante una estructura de datos de 3 componentes.Con la existencia de este nuevo recurso para el manejo de los datos, se acentúa la importancia de orientar

el pensamiento, en la etapa de análisis del problema, hacia la identificación de la información que haya que tratar. Tenemos que hacer una representación intelectual de los datos y su organización. Será también sumamente relevante reconocer el tipo de valores que se manejarán (si son números, caracteres,. . . .) y pensar e imaginar cómo disponerlos en memoria para que resulten adecuados a las necesidades del proceso de solución del problema.Cuando los datos se organizan en memoria como una estructura, existe la posibilidad de acceder a cada integrante del conjunto para procesarlo en forma individual (modificar su valor, mostrarlo, usarlo para algún cálculo o proceso particular, etc.). Para ello, cada valor podrá ser "referenciado" independientemente, según determinadas reglas (que estudiaremos luego para cada tipo de estructura) que son, en general, propias de cada estructura y en particular propias del lenguaje con que se codifique el algoritmo.El concepto de estructura de datos que hemos descrito aplicado a la memoria principal, se usará también para el almacenamiento masivo de datos en la memoria secundaria, aunque este tema será específicamente tratado en el capítulo que trata el tema Archivos.

6.0.2 Porqué utilizar estructuras de datos?

La estructura de datos es un recurso que permite operar con un grupo de valores que, en el contexto de una determinada situación problemática, tiene significado como tal. No en todos los casos en que intervengan grupos de datos se justifica el uso de estructuras de datos; elegirlas es una decisión importante, ya que ello influye de manera decisiva, como veremos más adelante, en el algoritmo que vaya a usarse para resolver lo que se requiera.
Por ejemplo, si del contexto de un problema se extrae que es necesario procesar el estado meteorológico de un día determinado en una ciudad (temperatura ambiente, humedad, precipitación), se podría utilizar, por ejemplo, una variable diferente para almacenar el valor de cada dato que conforma el estado meteorológico:
real: **temp_A, hum_A, lluv_A**

En cambio, si del contexto del problema surgiera que es necesario procesar también el estado meteorológico de otro día de la semana en esa ciudad; habría que disponer de otras tres variables completamente independientes de las demás, por ejemplo:real: **temp_A, temp_B, lluv_B, lluv_A, hum_A, hum_B**

Para usarlas de manera correcta el diseñador del algoritmo debe recordar cuál variable corresponde a un día y cuál al otro. Le resultaría más complicado aún, si tuviera que diseñar la gestión del estado meteorológico de varios días, como serían los de una semana o de un mes.Lo mismo podríamos decir de otras unidades de información tales como:

- los datos personales de un estudiante, que podría incluir apellido, legajo, edad, carrera que cursa.
- un número complejo, que comprende a su parte real y su parte imaginaria.
- una fecha, que comprende día, mes y año expresados como tres números o un número, una palabra y otro número.
- las alturas máximas mensuales del río Paraná, registradas en un lugar durante un año calendario, que incluye 12 alturas máximas.
- el nombre de una ciudad, que está formado por una definida cantidad de letras.

En todos los casos, en que la unidad de información a procesar sea un conjunto de datos, se puede recurrir al uso de un dato estructurado para representarlo.Evidentemente, este esquema lógico no es rigurosamente previsible fuera del contexto de la situación problemática; está, como dijimos antes, estrictamente ligado a ella. Por este motivo, muchas veces la dificultad para diseñar un

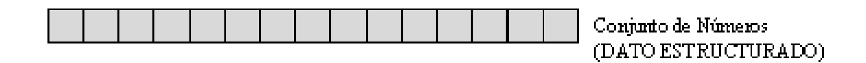
algoritmo radica en la elección de la/s estructura/s de datos adecuada/s. No podemos hablar de "reglas preestablecidas" para ello, todo se sustenta en el análisis que se haga del problema, en el conocimiento que se tenga de las características de las distintas estructuras de datos y en la experiencia adquirida por el uso de las mismas.

6.0.3 Ejemplos de usos de estructuras de datos

A continuación se plantean distintas situaciones problemáticas que servirán para ejemplificar en cuáles circunstancias es adecuado elegir una estructuras de datos para organizar la información a procesar y en cualés no. Se hace un breve análisis y se representan gráficamente, en forma esquemática, los lugares de memoria que ocuparían los valores a procesar; no nos ocuparemos de cómo llevar a cabo el proceso.
Si bien la memoria no está organizada según un patrón geométrico, imaginaremos con una figura de forma rectangular, al lugar que en ella ocuparía cada valor a mantener. De esta forma, asociaremos el espacio de memoria dispuesto para una estructura de datos con un agrupamiento de rectángulos contiguos (cada uno para un componente de la estructura).En realidad, los valores están dispersos en un área amplia de la memoria, pero los lenguajes de programación están diseñados para que el programador los pueda manipular "como si estuvieran" acomodados en forma adyacente.En la etapa de desarrollo del algoritmo es suficiente que nos limitemos a considerar la posibilidad de trabajar con grupos de datos organizados de esta manera ya que es, básicamente, la forma a la que se ajustan los lenguajes de programación que están a nuestro alcance. Cuando se estudie un lenguaje, será necesario aprender a usar las herramientas específicas de que se dispone para expresar el tratamiento de las estructuras de datos que se han planteado desde la algoritmia.

1° *Se dispone de 15 números enteros. Mostrarlos ordenados en forma creciente.*
Datos: cada uno de los números de un conjunto finito (son 15 números)
Resultado: el conjunto de números, clasificados de menor a mayor.
Proceso: ordenar los números del conjunto.

Nos preguntamos ¿cómo se pueden usar los recursos de una computadora para llevar a cabo este proceso?. Intuitivamente podemos pensar que la computadora tendrá que disponer del conjunto completo de números en su memoria, para poder compararlos entre si y decidir la nueva ubicación de cada uno ellos.Por lo tanto, la unidad de dato a procesar es el conjunto de números y los lugares de memoria que se requerirían para guardar a todos esos números, se podría representar gráficamente como:



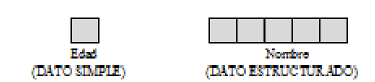
2° *Se dispone de 15 números enteros. Mostrar el resultado de sumar a todos los que pertenecen al [-5, 38].*
Datos: cada uno de los números de un conjunto finito (son 15 números)
Resultado: un único valor, resultante de una sumatoria.
Proceso: sumar sólo aquellos números del conjunto dado, que cumplan una determinada condición.

Para sumar los números enteros que sean mayores a -5 y menores a 38, no es indispensable tener

el conjunto completo de números en memoria, bastaría con tenerlos y procesarlos de a uno por vez.Por lo tanto, la unidad de dato a procesar es un número y la unidad de dato resultante del proceso es otro número.La representación gráfica de los lugares de memoria a usar sería:

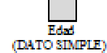


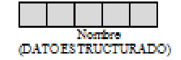
3°*Se dispone de la edad de una persona y de su nombre. Se quiere saber si se trata de un menor de edad (menos de 18 años) y si el nombre es JUAN.*
Datos: edad y nombre de una persona, valor de edad máxima y nombre determinado (18 y JUAN respectivamente)Resultado: dos mensajes. Cada uno de ellos aseverará o no, sendas afirmaciones que se plantean en el enunciado.Proceso: comparar el valor de cada dato de la persona con el correspondiente dado.En este caso hay 2 unidades de datos a procesar: la edad y el nombre de una persona (el primero es un dato simple y el segundo es un conjunto de caracteres).
La representación gráfica de los lugares de memoria a usar sería:



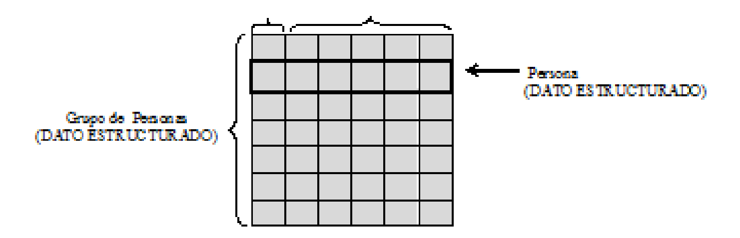
4°*Se dispone de la edad y el nombre de 7 personas. Determinar cuántos son menores de edad (menos de 18 años) y cuántos se llaman JUAN.*
Datos: edad y nombre de cada una de las personas de un conjunto finito (son 7 pares de datos), valor de edad máxima y nombre determinado (18 y JUAN respectivamente)Resultado: dos valores. Cada uno corresponde a la cantidad de veces que se verifica determinada afirmación.Proceso: contar la ocurrencia de algo. Para ambos casos lo que debe contarse es la veracidad de una comparación (valor de la edad comparada con 18, valor de nombre comparado con JUAN).La situación descrita en este problema es, en síntesis, igual a la anterior. El proceso al que debe someterse a los valores individuales de edad y nombre es la comparación de cada uno de ellos con otro valor, que está dado. La diferencia radica en lo que hay que hacer con el resultado de esa comparación. Por eso, en este caso, la respuesta definitiva se alcanza luego de procesar los valores de edad y nombre de las 7 personas (en el anterior era sólo de una).El tratamiento de los datos de cada persona (edad y nombre), es independiente de la cantidad de las mismas (sea una o más). Por lo tanto, la computadora NO necesita disponer del conjunto completo de datos en su memoria; bastará con los que correspondan a una persona por vez.Hay entonces, 2 unidades de datos a procesar: la edad y el nombre de cada una de las persona (el primero es un dato simple y el segundo es un conjunto de caracteres).La representación gráfica de los lugares de memoria a usar sería:

5°*Se dispone de la edad y el nombre de 7 personas. Mostrar el nombre de los que tienen más edad que el promedio del grupo.*
Datos: edad y nombre de cada una de las personas de un conjunto finito (son 7 pares de datos)

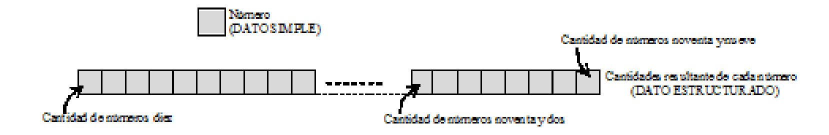




Resultado: uno o más valores (no podemos precisarlo a priori).
Proceso: calcular el valor promedio de uno de los datos (edad) y luego comparar ese promedio con cada uno de los valores que se usó para obtenerlo; de esa comparación resultará la vinculación o no con el otro dato (nombre).Evidentemente hay un dato (edad), cuyos valores serán procesados de distinta forma en distintos momentos: primero obteniendo la edad promedio y luego comparando cada edad con ese resultado. Pero no debe olvidarse, que a cada dato que interviene en el cálculo del promedio le corresponde otro dato (nombre) cuyo valor podrá ser o no el resultado del problema, dependiendo esto de la comparación mencionada (edad promedio ~edad individual).En este caso, la computadora tendrá que disponer del conjunto completo de los datos de las personas en su memoria, para poder realizar todo el tratamiento, pues éste requiere de un mismo valor más de una vez (edad).Hay un conjunto de datos a procesar: todo el grupo de personas (es un conjunto formado por edad y nombre).La representación gráfica de los lugares de memoria a usar sería:



6º*De un conjunto de números enteros positivos de dos cifras con repeticiones, se quiere determinar la cantidad de veces que aparece cada número, en el conjunto.*Datos: cada uno de los números de un conjunto finito (no hay información acerca de la cantidad de números que integran el conjunto). El valor de cada número es cualquiera que pertenezca al [10, 99].Resultado: noventa valores. Cada uno corresponde a la cantidad de veces que un valor se repite en el conjunto.Proceso: contar las veces que cada número se repite en el conjunto (cantidad de 10, cantidad de 11,. . . , cantidad de 99).En este caso, la unidad de dato a procesar es cada número del conjunto.De las características del proceso a que se debe someter a cada número (contarlo según su valor) resulta que NO es necesario mantener en memoria a todos ellos; aunque SI será indispensable que permanezcan disponibles en memoria, 90 lugares distintos. Cada lugar deberá ser usado para contar la ocurrencia de cada número. En definitiva, habrá un conjunto de datos resultante.Representación gráfica de los lugares de memoria a usar:*



6.0.4 Clasificación de las estructuras de datos

Uno de los criterios por los que se puede clasificar las estructuras de datos organizadas en memoria principal, tiene que ver con la posibilidad o no de variar su tamaño en algún paso del algoritmo.

- **Estructuras estáticas:** conservan su tamaño, desde que empieza hasta que termina el algoritmo que las usa. El espacio que ocupan en memoria se fija antes que se ejecuten las acciones y no existe ninguna, que le pueda ordenar al procesador alterar la cantidad de componentes, se los use o no.

El tamaño de las estructuras estáticas no puede modificarse una vez definida, aunque no necesariamente tiene que utilizarse toda la memoria reservada al inicio.

Hay distintas estructuras estáticas, cada una de las cuales responde a un formato general específico. Vamos a estudiar algunas de ellas, que son usadas en todos los lenguajes de programación: Registro, Arreglo y Cadena.- **Estructuras dinámicas o flexibles**: su tamaño puede crecer o decrecer, durante la ejecución del algoritmo que las usa. El espacio que ocupan en memoria NO puede ser predeterminado, depende de lo que se vaya necesitando para satisfacer los requerimientos de la aplicación. Estas estructuras pueden soportar acciones especiales que, durante la ejecución, permiten disponer de memoria adicional para ella o, por el contrario liberar la que se le haya asignado.

El tamaño de las estructuras dinámicas podrá modificarse; teóricamente no hay límites a su tamaño, salvo el que impone la memoria disponible en la computadora.

Entre estas estructuras podemos mencionar: Lista, Árbol, Grafo, etc.

Las estructuras de datos pueden ser clasificadas también, atendiendo al tipo de dato de sus componentes.

- **Estructuras homogéneas**: organizan un conjunto de datos de igual tipo. Entre éstas encontramos a los Arreglos y a las Cadenas (string).
Las podemos imaginar y representar gráficamente con cualquiera de los siguientes esquemas:

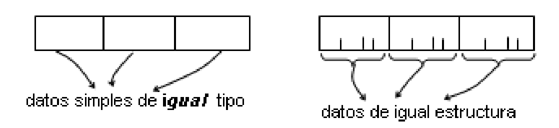


Fig 1 Representación gráfica de datos estructurados homogéneos

- **Estructuras heterogéneas**: organizan un conjunto de datos de distinto tipo. Entre éstas encontramos a los Registros. Las podemos imaginar y representar gráficamente con el siguiente esquema:

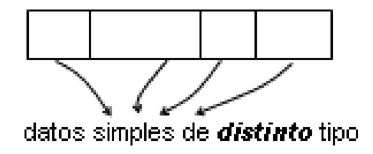


Fig 2 Representación gráfica de un dato estructurado heterogéneo

Tipo de dato estructurado

Los lenguajes de alto nivel, para solucionar el problema de las distintas representaciones internas de valores y de la organización de los datos correspondientes, nos brindan la posibilidad de encuadrar los datos a procesar en un modelo preestablecido. En un sentido amplio, esto implica imponer restricciones referidas a qué valores pueden tomar los datos, qué operaciones se pueden realizar y qué propiedades los caracterizan. Por ejemplo: al declarar una variable M numérica, se está determinando de qué forma se almacenarán sus posibles valores en cuanto a cantidad de bytes y a la disposición de los bits en ellos (sea según la notación en punto fijo o en punto flotante) y se está definiendo que la suma y la resta, entre otras, son operaciones a las que se podrán someter los valores de M, no así la concatenación, pues no está definida para los números.

Así es como los lenguajes definen a los tipos de datos. Casi todos incluyen en forma explícita, una determinada notación para los distintos tipos correspondientes a datos simples, aunque no todos usan la misma sintaxis (son los llamados tipos fundamentales, primitivos o estándares).Por otro lado, estos mismos lenguajes, permiten al programador “inventar” tipos de datos adicionales, normalmente combinando múltiples elementos de otros tipos. Se los denominan en forma genérica como tipos de datos definidos por el usuario.Tradicionalmente, bajo esta denominación los lenguajes tienen a una herramienta apropiada solamente para definir nuevos sistemas de almacenamiento. No llegan a crear realmente, en toda su magnitud, un nuevo tipo de dato, aunque así se los nombre, ya que estos lenguajes no cuentan con mecanismos que permitan definir algún conjunto de operaciones a realizar con los valores del nuevo tipo. Por ejemplo: al declarar una variable Z de un tipo especial previamente definido, solamente se está determinando de qué forma se almacenarán los posibles valores de cada uno de sus componentes y no se está previendo ninguna operación para Z como conjunto.Si bien, actualmente los lenguajes contemplan la posibilidad de definir tipos de datos de una forma más completa, pues incluyen el sistema de almacenamiento y las operaciones asociadas, en este libro trataremos sólo la forma tradicional.Siendo las estructuras de datos un mecanismo sumamente útil para almacenar y procesar conjuntos de datos en la memoria principal, resultan completamente apropiadas para implementar con ellas algunos de estos tipos de datos particulares, especialmente diseñados para la aplicación que se esté realizando. Se trata del denominado tipo de dato estructurado o compuesto.

Un tipo de dato estructurado o compuesto es una forma general de describir bajo un único nombre, a una estructura particular, pudiendo representar con él, a un lugar de memoria capaz de alojar un conjunto de valores

Recordemos que, recién cuando se declara una variable se está identificando al lugar de memoria que puede almacenar valores con determinadas características.Entonces, si al diseñar la solución de un determinado problema, hubiera que manipular algún dato estructurado, habría que:

1º definir la estructura adecuada, identificándola como un tipo especial mediante un nombre particular. Por ejemplo: registro R, registro dat, cadena meses, arreglo LOS, etc. y se describirán sus componentes

2ºdeclarar una o más variables, según las necesidades, de ese nuevo tipo de dato. Por ejemplo: declarar zeta de tipo R significa identificar con el nombre zeta a un lugar de memoria que puede almacenar valores con las características de las componentes que se hayan definido para R; zeta es una variable compuesta o estructurada.Algunos lenguajes de programación permiten la definición de tipos de datos estructurados, sólo para algunas estructuras, no para todas. En el caso que, con la estructura que se necesite usar, no pueda definirse un tipo de dato especial, se omitirá el 1ºpaso pero, la descripción de la estructura se incluirá en la declaración de la variable correspondiente al dato estructurado; y esto se repetirá para una o más variables, según se necesite. Retomaremos luego estos conceptos, en forma más detallada y específica, al describir las distintas estructuras de datos.

6.0.5 Estructuras de datos estáticas

Como se vio, se trata de estructuras que no pueden crecer ni achicarse. Pero, a pesar de esto, puede lograrse que en el conjunto de datos contenido en ella, "se agreguen" o "se quiten" valores. En algunos casos esto se conseguirá mediante el uso de funciones internas especialmente diseñadas y en otros, a través del desarrollo de algoritmos específicos.Como se vio en **Ejemplos de usos de estructuras de datos**, del análisis de la situación problemática planteada surgirá la conveniencia de manipular los datos almacenándolos en un formato simple o como una estructura. En este último caso, aparecerá inmediatamente el interrogante de ¿cuál elegimos, de todas las posibles estructuras?. Fundamentalmente la toma de decisión dependerá del conocimiento que se tenga de las características de cada una de ellas pues de esa forma se podrá seleccionar la más adecuada. De cualquier manera también tendrá peso en la elección, aunque en menor medida, las herramientas que brinde el lenguaje de programación que se vaya a utilizar en la codificación para implementar las estructuras.A continuación comenzaremos el estudio de varias estructuras de datos, considerando a tres organizaciones, las denominadas cadena, registro y arreglo.Como nuestra tarea se circunscribe al desarrollo de algoritmos, acordamos que, sólo con la estructura de datos registro se podrá implementar un tipo de dato; mientras que no se hará así con la cadena y el arreglo, por lo tanto la estructura de estos últimos se definirá al declarar la variable correspondiente.Al codificar el algoritmo en un determinado lenguaje (no se hará en este libro) habrá que adecuar el manejo que se haya planteado para las estructuras de datos usadas, con las limitaciones semánticas del lenguaje.

6.1 Cadena (String)

6.1.1 6.1.Estructura de datos: Cadena (string)

Cuando sea necesario tratar como unidad a un dato, cuyo valor estará formado por una sucesión de sólo caracteres alfanuméricos (por ejemplo: la matrícula de un vehículo, la frase escrita por alguien, el nombre de una sustancia, etc.), estamos en presencia de un dato compuesto especial. La cadena o también llamada string, es la estructura apropiada para contenerla. Por eso definimos:

Una cadena es una estructura de dato que puede tomar por valor una sucesión de caracteres

Esta estructura es estática, porque el espacio de memoria que se reserva para todas sus componentes no puede modificarse una vez que se lo haya fijado y es homogénea, porque todos sus componentes son de un único tipo, el carácter. Se trata de cualquiera que pertenezca al juego de caracteres que utiliza el lenguaje de programación que se use para codificar el algoritmo.Uno de

los códigos más utilizados es el ASCII (American Standard Code for Information Interchange) que consiste en 256 símbolos, entre los que se encuentra el espacio en blanco, las letras del abecedario, los símbolos numéricos, los signos de puntuación, otros símbolos especiales y algunos caracteres que no pueden ser mostrados como tales, porque efectúan una operación específica sobre algún dispositivo (es el caso del retorno de carro, del salto de página, etc.). Recordamos, a continuación, algunos de los caracteres tabulados en ASCII:

ASCII DE CARACTERES NO IMPRIMIBLE			ASCII DE CARACTERES IMPRIMIBLES						
⋮	⋮		⋮	⋮	⋮	⋮	⋮	⋮	
07	*BEL*	*Timbre*	32	*espacio*	88	*X*	162	*ó*	
08	*BS*	*Retroceso*	33	*!*	89	*Y*	163	*ú*	
09	*HT*	*tab horizontal*	34	*"*	90	*Z*	164	*ñ*	
10	*LF*	*nueva línea*	35	*#*	91	*[*	165	*Ñ*	
11	*VT*	*tab vertical*	36	*$*	92	\	166	ª	*ordinal femenino*
12	*FF*	*nueva página*	37	*%*	93	*]*	167	º	*ordinal masculino*
13	*CR*	*retorno de carro*	38	*&*	94	^	168	*¿*	
⋮		⋮	⋮	⋮	⋮	⋮	⋮	⋮	
24	*CAN*	*Cancelar*	47	*/*	121	*y*	246	÷	*signo de división*
25	*EM*	*fin del medio*	48	*0*	122	*z*	247	¸	*cedilla*
26	*SUB*	*Sustitución*	49	*1*	123	*{*	248	°	*signo de grado*
27	*ESC*	*escape*	50	*2*	124	|	249	¨	*diéresis*
⋮	⋮		⋮	⋮	⋮	⋮	⋮	⋮	
127	*DEL*	*suprimir*	64	*@*	136	*ê*	255	*nbsp*	*non breaking space (espacio sin separación)*

Por ejemplo, algunos valores que pueden ser almacenados en cadenas son: ‘La manzana está muy verde’, ’12,35’, ¿ &%$xX ¬@’, ‘1027’, ‘Le bon élève’.Con el término cadena identificamos a la estructura en memoria pero también, usaremos esa palabra cuando nos expresamos en forma coloquial, como sinónimo de “conjunto de caracteres”, es decir para referirnos al valor alojado en ella.La longitud de una cadena es la cantidad de caracteres que la integra. Una cadena vacía o nula es la que no contiene caracteres y por lo tanto su longitud es cero.Cuando se defina una estructura de datos cadena, sólo deberá tenerse en cuenta que habrá que prever un tamaño mayor o a lo sumo igual al del posible valor y, cuando la secuencia de caracteres se aloje en la estructura, no habrá que preocuparse por indicar el “fin de la cadena”, salvo que, por las exigencias del problema, sea esto estrictamente necesario. Consideraremos que el procesador, de alguna manera, sin intervención del programador, “se encargará” de “marcar” el fin de la secuencia.Por todo esto, en los ejemplos anteriores: ‘La manzana está muy verde’ es una cadena de 25 caracteres y ‘12,35’ es una cadena de 5 caracteres. Cualquiera de los dos valores puede contenerse en memoria en una cadena definida con un tamaño de 30, no así, si se la definiera de 7 (sólo serviría para el ‘12,35’).La disposición de la cadena en la estructura, la podemos representar como en las siguientes figuras:

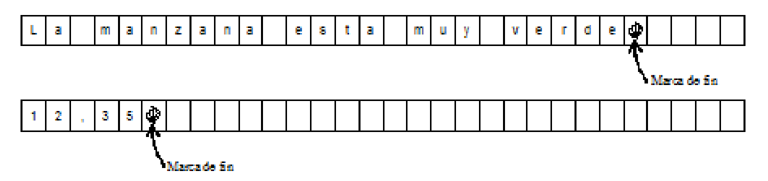


Fig 3 Cadenas de caracteres dispuestas en memoria

Si bien no todos los lenguajes de programación manipulan esta estructura de datos de igual forma, de una manera u otra, todos cuentan con alguna herramienta para implementarla en memoria:
- hay lenguajes que NO incluyen a la cadena como estructura de datos; por lo tanto usan otra estructura (su descripción se verá más adelante en este capítulo) para implementar en memoria un dato cadena (C/C++)- hay lenguajes que SI incluyen a la cadena como estructura de datos pero NO permiten definir un TIPO DE DATO con ella; por lo tanto la estructura se la define cuando se declara una variable con sus características (FORTRAN)- hay lenguajes que SI incluyen a la cadena como estructura de datos pero incorporada como TIPO DE DATO CADENA (generalmente con el nombre de string o String) y se lo usa como si fuera un tipo de dato estándar más; entonces el programador sólo definirá el tamaño de la cadena y declarará variables de ese tipo (PASCAL).Como ya se mencionó adoptaremos una forma genérica para el tratamiento de las cadenas, definiendo la estructura en el momento de la declaración de la variable correspondiente.

6.1.2 Declaración de una variable cadena

Notación algorítmica en pseudocódigo:

- *caracter (cantidad de caracteres): Nombre de la variable*

Ejemplos:
caracter(18): NM, DL reserva lugar para dos variables, denominadas NM y DL respectivamente. Cada una puede contener hasta 18 caracteres.caracter(5): x reserva lugar para una variable denominada x que puede contener hasta 5 caracteres.

6.1.3 Usos de una variable cadena

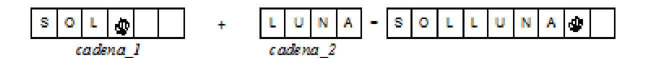
A las cadenas se las trata como a cualquier dato simple. Pueden ser usadas en acciones primitivas (leer, asignar, escribir), como operando de expresiones o como argumentos de funciones.Ejemplos de primitivas aplicadas a las variables anteriores:

1. NM ← "Villa del Sur"
2. leer (x)
3. escribir (DL)

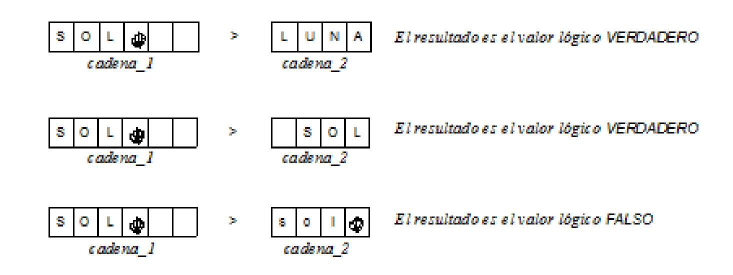
6.1.4 Operaciones con cadenas

Los lenguajes brindan la posibilidad de realizar muchas operaciones sobre cadenas, mediante el uso de algunos operadores o de funciones internas que las tienen como argumento o que las devuelven como resultado. Entre uno y otro lenguaje suele haber diferencias, en cuanto a la variedad de operaciones permitidas, a la sintaxis requerida para ellas y a la mayor o menor facilidad que ofrecen para su aplicación en una acción algorítmica.
Presentamos a continuación, en pseudocódigo, las operaciones básicas más comunes que todos los lenguajes permiten poner en práctica. En el caso de los operadores, empleamos símbolos estándares y para las funciones utilizamos un nombre genérico que permita distinguir lo que la función hace sobre la cadena.~Concatenación (operador +): se unen dos o más cadenas para resultar otra.
cadena_1 + cadena_2

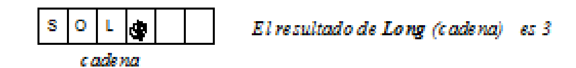
~Comparación (un operador relacional): se vinculan cadenas como operandos de la expresión



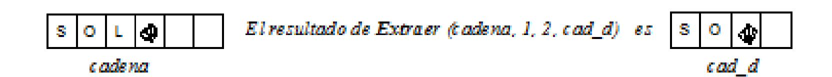
relacional. La comparación responde a la secuencia normalizada en la tabla ASCII. Por lo tanto, una cadena es mayor que otra cuando sus caracteres son posteriores, en la tabla, a los caracteres de la otra cadena. En el caso especial de los caracteres alfabéticos, resulta ser el orden creciente del abecedario, diferenciándose las mayúsculas de las minúsculas. cadena_1 > cadena_2



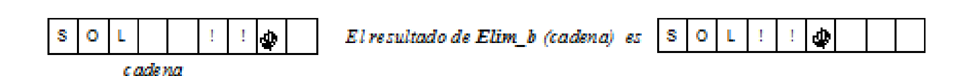
~Cálculo de la longitud: se cuenta la cantidad de caracteres que la forman.
Long (cadena)



~Extracción de subcadena: se "recorta" una cadena para, con ello, formar otra.
Extraer (cadena_origen, posición inicial, posición final, cadena_destino)



~Eliminación de los espacios en blanco: se "compacta" la cadena quitándole todos los espacios finales para, con ello, formar otra.
Elim_b (cadena)



Ejemplos:

1. NM ← “Villa” + “ del ” + “Sur”
2. TOT ← Long (NM)
3. si (DL > “*150#”) entonces ......
   fin_si

6.2 Registro (Record o Struct)

Cuando la unidad a procesar es un dato compuesto que está formado por datos de distinto tipo, la estructura apropiada para contenerlo es el registro. Definimos así:

Un registro es una estructura de datos que puede tomar por valor a un conjunto de distintos o iguales tipos de datos

Esta estructura es estática, porque el espacio de memoria que se reserva para todos sus componentes no puede modificarse una vez que se lo haya fijado, y es heterogénea porque sus componentes pueden ser de diferentes tipos, aunque esto no invalida que puedan ser de igual tipo.Cada componente o elemento de un registro se llama campo o miembro y se identifica por un nombre.Los campos de la estructura se corresponden con lugares adyacentes de memoria y su funcionalidad es la misma que la de una variable. Por lo tanto, trabajaremos con el campo de un registro de la misma forma que lo haríamos con cualquier variable. Es importante tener en cuenta, que los campos no están restringidos a datos simples; por el contrario, podrían estar constituidos por cualquier estructura de datos válida.Ya vimos que del análisis minucioso de la situación problemática, surgirá el o los datos a tratar (ejemplos presentados en la parte inicial del capítulo en **¿Por qué utilizar estructuras de datos?**. Cuando **é**stos sean compuestos, habrá que observar detalladamente sus características para definir la estructura apropiada.
Podemos pensar, por ejemplo, que son datos compuestos que pueden implementarse con un registro:

- un cheque (banco emisor, importe, fecha),
- una ficha de empleado (legajo, apellido y nombre, D.N.I., cargo que desempeña)
- los datos meteorológicos de un lugar (hora, temperatura, humedad, presión)
- la medida de un ángulo expresada en el sistema sexagesimal (grados, minutos, segundos)
- un alumno de Informática I (legajo, apellido y nombre, edad, carrera)
- una fecha (número del día, nombre del mes, número del año)
- una fecha (número del día, número del mes, número del año)

En cambio, NO sería adecuado pensar en implementar con un registro, a los siguientes datos compuestos:

- el personal administrativo de un negocio (son varios empleados)
- una comisión de Informática I (son varios alumnos)
- las fechas de los feriados de este año (son varias fechas)
- los meses del año (son 12 palabras)
- los números primos de dos cifras (son varios números enteros, aunque seguramente no serán más de 90)

Como ya se mencionó, para utilizar un registro habrá que definir un tipo de dato general y luego reservar lugar de memoria mediante la declaración de una variable de ese tipo especialmente creado.

6.2.1 Definición de un tipo de dato registro

Definir un tipo de dato registro es establecer la organización de la estructura bajo un nombre expresamente elegido (Nombre del nuevo tipo).
Se comienza con la denominación de la estructura correspondiente (registro) y a continuación se describen los campos que la constituyen (identificador y tipo de dato de cada uno) en el orden que tienen en el conjunto, con lo que queda fijado el tamaño del bloque de memoria que ocupará la estructura.Notación algorítmica en pseudocódigo:

TIPO
registro Nombre del nuevo tipo
tipo de dato: Nombre 1er.campo
tipo de dato: Nombre 2do.campo
....
tipo de dato: Nombre último campo

Con la palabra **TIPO** se está indicando que lo que sigue, es la definición de un tipo de dato.

Ejemplos:
TIPO
***registro** dato_alum*
caracter(30): m
real: n
lógico: p

***registro** fecha*
entero: dia
entero: mes
entero: anio

***registro** cheque*
caracter(22): bco
real: imp
fecha: fe
caracter(30): ap_nom

TIPO
***registro** tiempo*
real: second
entero: minut
entero: hs

Con las palabras dato_alum, fecha y cheque se ha elegido identificar, en un determinado algoritmo, a tres formatos distintos de registro.
Con la palabra tiempo se ha elegido identificar, en un determinado algoritmo, a un formato de registro.

6.2.2 Declaración de una variable de tipo de dato registro

Sólo después de definidos el o los tipos de datos, se podrán declarar variables que respondan a algunas de esas organizaciones. Nunca podrán ser declaradas de un formato que no se haya establecido antes como tipo.
Notación algorítmica en pseudocódigo:

- Nombre del tipo de dato registro: Identificador de la variable

Donde:
Nombre del tipo de dato registro: será un nombre cualquiera que el programador elegirá para el nuevo tipo de dato.

Ejemplos: Se declaran variables de tipos de datos definidos en el ejemplo anterior:

tiempo: x reserva un lugar en memoria con el nombre x. En él se podrá alojar el valor de un dato

- compuesto que tenga primero un número fraccionario y después dos enteros. Nunca otros tipos de valores.

dato_alum: pers, uno reserva dos lugares en memoria, denominados pers y uno, respectivamente.

- Los dos pueden alojar a los valores de un dato compuesto que tenga primero una cadena de hasta 30 caracteres, después un número fraccionario y finalmente un valor lógico. No servirán para contener a los valores de un dato compuesto que tenga, por ejemplo, primero un número fraccionario, después un valor lógico y finalmente una cadena de hasta 30 caracteres.

Evidentemente en todas las variables que se declaren de un mismo tipo registro, la identificación de los campos será siempre mediante el nombre que se estableció en la definición del tipo. Estos nombres no interferirán entre una y otra variable. Así, pers y uno pueden coexistir en memoria y sus campos tendrán la misma identificación, pero corresponderán a lugares distintos; el 2do. campo de pers es n y el 2do. campo de uno es n.Si las declaraciones anteriores formaran parte de un mismo algoritmo, podríamos representar la memoria como:

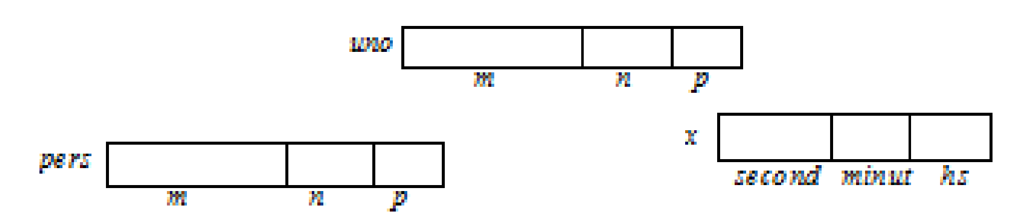


Fig 4 Representación de una memoria al declararse 3 registros

6.2.3 Acceso a un campo de la variable registro

Para acceder a un campo, bastará con mencionar su identificador y el nombre del registro del que forma parte. Se usa un operador especial para separar ambos nombres; en el contexto de este libro el símbolo será el punto, aunque en otros entornos suele usarse el signo de porcentaje.
Notación algorítmica en pseudocódigo:

- Identificador de la variable registro . Identificador del campo

Ejemplos: Se usan variables declaradas en el ejemplo anterior:

x.hs referencia al 3er. campo del registro x.

uno.m referencia al 1er. campo del registro uno.

pers.m referencia al 1er. campo del registro pers.

pers.p referencia al 3er. campo del registro pers.

6.2.4 Usos de una variable registro

Los registros no se tratarán en bloque. Se lo hará campo a campo, cada uno en forma independiente. Se trabajará con el campo de la misma forma que se lo haría con cualquier variable de su tipo. Por lo tanto, son los campos del registro los que podrán ser usados en acciones primitivas (leer, asignar, escribir), como operando de expresiones o como argumentos de funciones.
Ejemplos: Se usan variables declaradas en ejemplos anteriores:

1. pers.m ← “Profesor”
2. leer (pers.n)
3. mientras (pers.p = .Verd.)
4. ....
5. fin mientras
6. escribir (pers.m)

6.2.5 Ejemplo integrador de conceptos referidos a registros

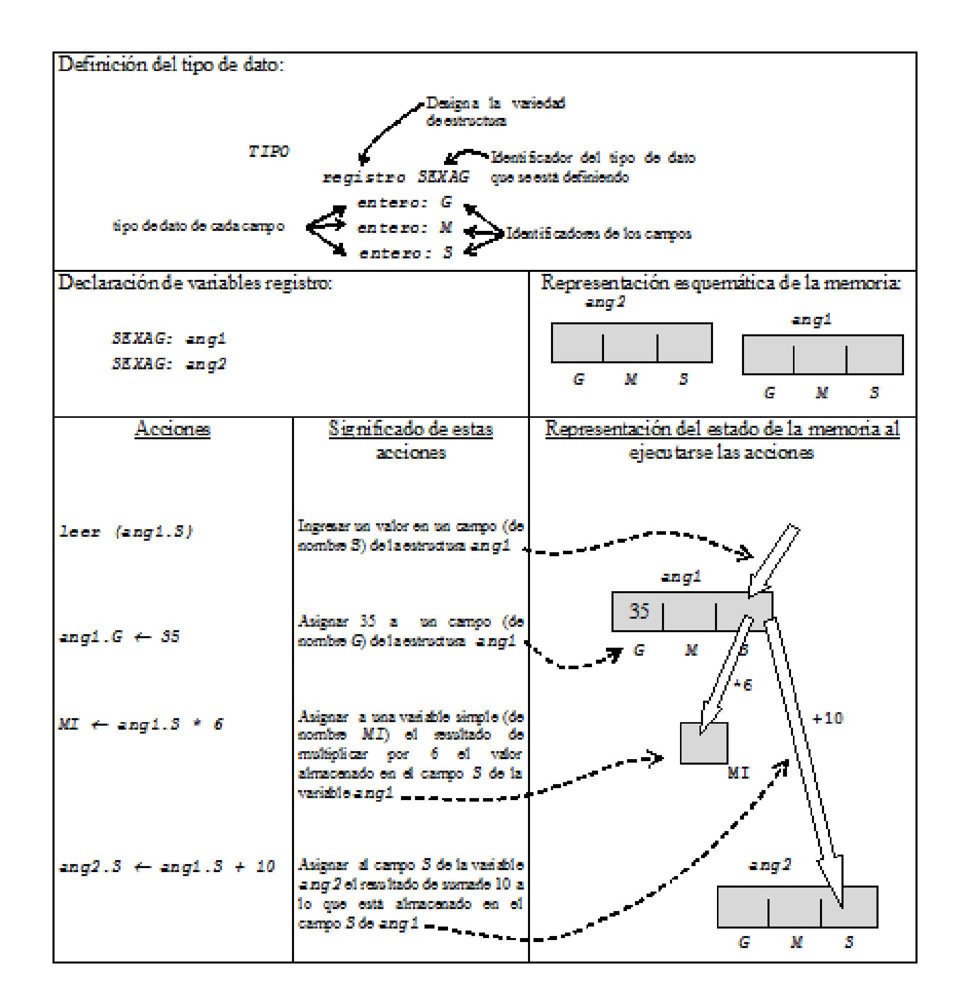
Para ejemplificar el uso de registro tomamos como punto de partida que, como consecuencia del análisis del enunciado del problema, se ha detectado un dato compuesto a procesar. Desde esta situación, realizamos:

1ºun análisis de las características del dato, para resolver o no su implementación con un registro.
2ºuna descripción de las acciones que podrían estar en el algoritmo y junto a ellas la representación de lo que ocasionan en la memoria.

Dato que se necesita procesar: Medida de un ángulo expresado en sistema sexagesimal.
Se trata de un dato compuesto, pues en el sistema sexagesimal la “medida de un ángulo” comprende indefectiblemente, a un conjunto de 3 números. No existe un tipo estándar que lo defina.Si suponemos que del análisis del enunciado surge la conveniencia de trabajar el dato como un todo, con la posibilidad de operar con sus componentes en forma individual, sería apropiado elegir organizarlos como un registro (de lo contrario podría ser factible usar 3 variables simples).

6.2.6 Otros ejemplos de datos que se podrían implementar con registros

Ejemplo 1)
Dato que se necesita procesar: Una fecha expresada en el formato DIAMESAAAA (nro. del día, nombre del mes, nro. del año), por ejemplo: 29 ABRIL 2006.



<u>Definición del tipo de dato para la fecha:</u>
TIPO
registro FECHA

- entero: dia
- caracter(10): mes
- entero: an

Ejemplo 2)
<u>Dato que se necesita procesar:</u> Un punto en el espacio (viene determinado por sus coordenadas).

<u>Definición del tipo de dato para un punto en el espacio:</u>
TIPO
registro PUNTO_3D

- real: X, Y, Z

Ejemplo 3)
<u>Dato que se necesita procesar:</u> Un vehículo de transporte.

<u>Definición del tipo de dato para un vehículo:</u>
TIPO

registro VEHICULO

- cadena(15): MARCA
- cadena(20): MODELO
- real: COSTO
- entero: ANIO
- lógico: DIESEL

6.3 Arreglo (Array)

Cuando la unidad a procesar es un dato compuesto que está formado por datos de igual tipo, la estructura apropiada para contenerlo es el arreglo. Definimos así:
Esta estructura es estática, porque el espacio de memoria que se reserva para todas sus componentes no puede modificarse una vez que se lo haya fijado, y es homogénea porque sus componentes son de igual tipo.Los arreglos se caracterizan por las siguientes propiedades:

1. Cantidad de componentes prefijada: esto determina el tamaño del bloque de memoria que ocupará.
2. Componentes directamente accesibles: al ubicarse en posiciones adyacentes son fácilmente individualizables.
3. Componentes de igual tipo: esto determina que, independientemente, todos ocupan la misma cantidad de bytes.

En forma simple lo podríamos graficar:

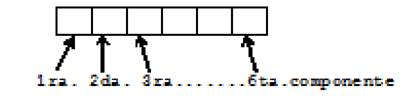


Fig. 5 Esquema representativo de un arreglo

Cada componente o elemento del arreglo se comporta como lo hace cualquier variable. Para identificarlo debe señalarse su posición relativa por medio de un índice (también llamado subíndice). Este irá tomando valores consecutivos a partir de uno dado como inicial.En este libro trabajaremos con índice numérico entero, comenzando por 1 (hay lenguajes de programación que aceptan otros valores como índice, tales como letras, otros rangos de números, etc.).Es necesario tener en cuenta que el valor de cada índice queda implícitamente definido al determinarse el tamaño del arreglo y nunca podrá ser modificado a lo largo del algoritmo.Como consecuencia de estas características, el dato compuesto que se implemente con un arreglo deberá identificararse por las siguientes propiedades:

- **conjunto finito**: está integrado por una determinada cantidad de valores que nunca será superior al tamaño que se fije para el arreglo que los contendrá.
Por ejemplo:

- NO se podrá recurrir a un arreglo para implementar:
  - los enteros positivos múltiplos de 3: ya que es imposible precisar su cantidad.
  - los importes de las ventas registrados de un comercio: en este enunciado no se precisa la cantidad de ventas.
- SI se podrá recurrir a un arreglo para implementar:

- los enteros positivos de 2 cifras que son múltiplos de 3: porque aún, sin realizar cálculos previos, podemos estar seguros que no serán más de 90 valores. Por lo tanto se podría pensar en un arreglo de un tamaño definido por exceso, en este caso de 90 componentes (aunque evidentemente no se ocuparán todos los lugares).
- los nombres de los meses del año (son 12).
- los importes de las ventas diarias totales de un comercio, registradas en el mes de diciembre (son, a lo sumo, 31 valores).
- la temperatura ambiente media diaria de 5 ciudades, durante una semana (son 35 valores)

- **conjunto de valores del mismo tipo**: pueden corresponder a datos simples (enteros, reales, lógicos, caracteres) o estructurados (cadena o registro).

Por ejemplo:

- los enteros positivos de 2 cifras, múltiplos de 3: son números enteros
- los nombres de los meses del año: son cadenas
- los importes de las ventas diarias totales de un comercio, registradas en el mes de diciembre: son números con parte fraccionaria.
- la temperatura ambiente media diaria de 5 ciudades, durante una semana: son números con parte fraccionaria.

- **conjunto ordenado:** cada dato ocupa en el conjunto, el lugar que lógicamente le corresponde, de acuerdo con la vinculación que los caracteriza y no con la "clasificación de sus valores". Esa posición es perfectamente identificable y es así que podemos referirnos al primero, al segundo, . . . , al enésimo integrante del conjunto.Por ejemplo:

- los nombres de los meses del año: enero es el 1ºmes del año, febrero el 2º, . . . , abril el 4º, . . . , diciembre es el 12º; este ordenamiento queda implícitamente definido al decir que son "los meses del año", no se ajusta a un criterio alfabético de las palabras que identifican a cada mes.

6.3.1 Disposición de los elementos

Dijimos que el arreglo, entre otras cosas, se caracteriza por ser una estructura que permite implementar conjuntos de datos cuyos elementos estén lógicamente ligados. De acuerdo con los aspectos que definen ese vínculo, pueden pensarse distintas maneras de distribuir los datos en la estructura. Conocer y estudiar esas diferentes formas, brinda una herramienta poderosa para decidir cuál es el arreglo adecuado a las características del dato compuesto que se necesita implementar.Teniendo en cuenta la forma con que los datos se organizan en la estructura, podemos clasificar a los arreglos en:

unidimensional o vector (se ajusta a un modelo lineal). Los datos se organizan como una lista, es decir acomodados en línea, uno al lado del otro. Cada componente se referencia por su ubicación en la lista, usándose para ello un solo índice. La cantidad de componentes constituye lo que se denomina rango del arreglo.

Entonces, gráficamente representamos:

Podría ser conveniente utilizar un arreglo unidimensional para:

- las calificaciones con que un profesional aprobó cada materia de su carrera (son tantas notas como materias tenga la carrera, siendo ésta una cantidad conocida)

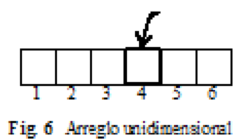


Fig. 6 Arreglo unidimensional (6 componentes)

- los valores de la temperatura en cada hora durante un día (corresponden a 24 mediciones)
- los coeficientes de un polinomio de grado 4 (son 5 números)
- los nombres, en castellano, de los días de la semana (son 7 nombres)

Cuando se describió a la estructura de datos cadena se mencionó que "lenguajes que NO incluyen a la cadena como estructura de datos; por lo tanto usan otra estructura…"; justamente, es un arreglo unidimensional de caracteres la estructura que permitiría implementar a un dato cadena. Por supuesto su tratamiento se ajustará al de un arreglo.
De acuerdo con lo ya descrito, quedará a criterio del programador usar el recurso que sea más apropiado para el proceso al que se necesita someter los valores.

bidimensional o matriz (se ajusta a un modelo matricial). Los datos se organizan como una tabla, es decir acomodados en un plano. Cada componente se referencia por dos índices, uno da la posición en sentido horizontal (fila) y el otro en sentido vertical (columna).

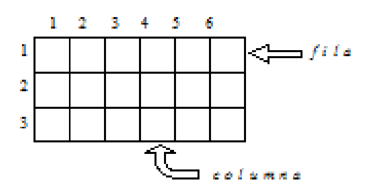


Fig. 7 Arreglo bidimensional (3 filas y 6 columnas)

En un arreglo de dos dimensiones a la cantidad de filas se la denomina rango de filas (para la Fig. 7 este valor es 3) y a la cantidad de columnas, rango de columnas (para la Fig. 7 este valor es 6).
Es así que se sintetiza el tamaño de la estructura con la expresión: rango de filas x rango de columnas.
El arreglo representado en la Fig. 7 es un bidimensional de 3 x 6. También se dice, de manera simplificada, que corresponde a un arreglo de dimensión 3 x 6 (al mencionar los dos rangos se está diciendo, implícitamente, que se trata de dos dimensiones).
En este libro, trataremos la referencia de cada componente por medio del par de índices ordenados como: n°de la fila, n°de la columna.

Entonces, gráficamente representamos:

Podría ser conveniente utilizar un arreglo bidimensional para:

- el importe de las ventas diarias de los vendedores de un negocio.

Se lo puede organizar con una columna para cada día y una fila para cada vendedor, ubicando en la intersección día-vendedor el importe de venta correspondiente. Aunque también se pudo haber organizado con una fila para cada día y una columna para cada vendedor.

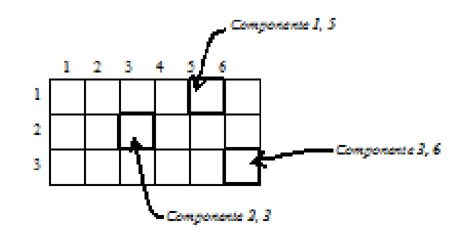


Fig. 8 Posiciones en un arreglo bidimensional

Ejemplo:
Dato: Importes de las ventas realizadas por 4 vendedores durante 3 días
Organización que puede imaginarse para este dato: arreglo de 4 x 3 o arreglo de 3 x 4

- los coeficientes de un sistema de ecuaciones de igual grado.
Se lo puede organizar con una columna para cada coeficiente de las ecuaciones y una fila para cada ecuación, ubicando en la intersección coeficiente-ecuación al correspondiente valor del coeficiente. Se pudo haber organizado con una fila para cada coeficiente y una columna para cada ecuación.

Ejemplo:
Dato: Coeficientes de un sistema de 5 ecuaciones de 2do. grado
Organización que puede imaginarse para este dato: arreglo de 5 x 3 o arreglo de 3 x 5

- una representación del estado de un tablero de ta-te-ti.
Se lo puede organizar con una columna por cada columna del tablero y una fila por cada fila del tablero, ubicando en una intersección fila-columna al estado de la casilla del tablero (un valor para la vacía, otro para la ocupada con ficha de un jugador y otro para la ocupada con ficha del contrincante).

Ejemplo:
Dato: Estado del tablero de ta-te-ti teniendo en cuenta que un jugador usa fichas 9679; y que el otro jugador usa fichas
Organización que puede imaginarse para este dato: arreglo de 3 x 3

- los nombres de los meses del año, expresados en idiomas distintos
Se lo puede organizar con una columna para cada idioma y una fila para cada mes, ubicando en la intersección idioma-mes al correspondiente nombre. Se pudo haber organizado con una fila para cada idioma y una columna para cada mes.

Ejemplo:
Dato: Nombres de los meses del primer trimestre del año en español, inglés, portugués y francés
Organización que puede imaginarse para este dato: arreglo de 3 x 4 o arreglo de 4 x 3

multidimensional: cada componente se referencia por tres o más índices.

Podría ser conveniente utilizar un arreglo tridimensional para:

- - la cantidad de alumnos inscritos en cada comisión de cada nivel de cada carrera de una facultad; supuestas todas las carreras con la misma cantidad de comisiones y de niveles.
- - la representación del estado de las ubicaciones de una sala de teatro para un determinado espectáculo; las localidades se identifican por un número de asiento dentro de una fila y en

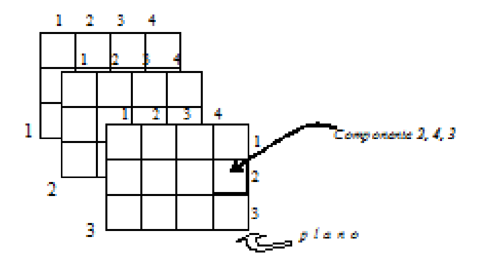


Fig 9 Arreglo tridimensional
(3 planos, cada uno con 3 filas y 4 columnas)

un nivel de la sala (un valor para localidad vendida y otro para localidad no vendida).
Una vez detectada la necesidad de procesar un dato estructurado con un arreglo es fundamental, para el buen manejo de la información, elegir el tipo de arreglo apropiado. Todo esto se hará a partir del análisis del problema.

6.3.2 Declaración de un arreglo

En la declaración debe quedar especificado un nombre o identificador para el arreglo, su tamaño y el tipo de datos de sus componentes.
Notación algorítmica en pseudocódigo:

- Tipo de dato Identificador del arreglo (tamaño)

Donde:
Identificador del arreglo: será cualquier nombre válido para una variable.
Tamaño: será el rango de cada dimensión del arreglo (separados por una coma)

Ejemplos:
1. entero M(20) arreglo de 20 números enteros
2. caracter(10) ZETA(5) arreglo de 5 cadenas de 10 caracteres cada una
3. dato_alum WIL(8) arreglo de 8 registros dato_alum antes definido
4. entero X(10,6) arreglo de 60 enteros organizados en 10 filas y 6 columnas

Una vez que se haya declarado el arreglo y por lo tanto, reservadas las posiciones de memoria, se podrán poner valores en la estructura.

Así, referido a los datos compuestos mencionados en algunos de los ejemplos anteriores, podríamos decir:

1)
Dato: Importes de las ventas realizadas por 4 vendedores durante 3 días
Valores posibles:

- Vendedor 4 $80 el 3er. día
- Vendedor 1 $ 9 el 2do. día
- Vendedor 3 $10 el 3er. día
- Vendedor 1 $5 el 1er. día

- Vendedor 1 $15 el 3er. día
- Vendedor 2 $30 el 1er. día
- Vendedor 3 $7 el 1er. día
- Vendedor 3 $18 el 2do. día
- Vendedor 4 $6 el 2do. día
- Vendedor 4 $20 el 1er. día
- Vendedor 2 $12 el 2do. día
- Vendedor 2 $90 el 3er. día

Estructura elegida: arreglo de 3 x 4
Arreglo organizado en memoria:

	1	2	3	4
1	5	30	7	20
2	9	12	18	6
3	15	90	10	80

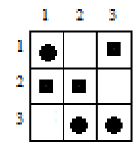
2)
Dato: Coeficientes de un sistema de 5 ecuaciones de 2do. grado
Valores posibles:

Ec. nº1: $3x^2 + 5x - 1$
Ec. nº2: $x^2 + 2{,}3x$
Ec. nº3: $6x^2 + x + 0{,}5$
Ec. nº4: $-7x^2 + 5x + 2$
Ec. nº5: $0{,}2x^2 + 1{,}8x - 3$

Estructura elegida: arreglo de 5 x 3
Arreglo organizado en memoria:

	1	2	3
1	3	5	-1
2	1	2,3	0
3	6	1	0,5
4	-7	5	2
5	0,2	1,8	-3

3)
Dato: estado del tablero de ta-te-ti teniendo en cuenta que un jugador usa fichas 9679;, y que el otro jugador usa fichas
Valores posibles:



Estructura elegida: arreglo de 3 x 3
Arreglo organizado en memoria:
Es necesario codificar las fichas con un valor almacenable en el arreglo (número, carácter, valor lógico), elegimos designar con 0 a la casilla vacía, con 1 a la ficha y con 9 a la ficha 9679;

	1	2	3
1	1	0	9
2	9	9	0
3	0	1	1

4)
Dato: Nombres de los meses del primer trimestre del año en español, inglés, portugués y francés
Valores posibles:

- 1ºmes del año
  - Español: Enero
  - Inglés: January
  - Francés: Janvier
  - Portugués: Janeiro
- 2ºmes del año
  - Español: Febrero
  - Inglés: February
  - Francés: Février
  - Portugués: Fevereiro
- 3ºmes del año
  - Español: Marzo
  - Inglés: March
  - Francés: Mars
  - Portugués: Mar

Estructura elegida: arreglo de 3 x 4
Arreglo organizado en memoria:

6.3.3 Acceso a un elemento del arreglo

Para acceder directamente a un determinado componente bastará con mencionar el identificador del arreglo y su posición en él.

	1	2	3	4
1	Enero	January	Janvier	Janeiro
2	Febrero	February	Février	Fevereiro
3	Marzo	March	Mars	Março

Notación algorítmica en pseudocódigo:

- Identificador del arreglo (índice) ó Identificador del arreglo (índice fila, índice columna)

Donde:
Identificador del arreglo: nombre con el que se lo ha declarado.
Indice: está dado por una variable (valor que existe en memoria), una constante o una expresión

Ejemplos:

- A(M) referencia al elemento que, en el arreglo A, ocupa una posición dada por el valor que tiene la variable M (si en M hay un 2 será A(2)).
- A(5) referencia al elemento que, en el arreglo A, ocupa la posición 5.
- A(b + 3) referencia al elemento que, en el arreglo A, ocupa la posición dada por el resultado de b + 3 (si en b hay un 2 corresponde a A(5)).
- Z(lal,7) referencia al elemento que, en el arreglo Z, ocupa la intersección de la fila cuyo valor viene dado por la variable lal con la columna 7 (si en lal hay un 2 equivale a decir Z(2,7)).

Cada elemento del arreglo podrá ser sometido a todas las operaciones permitidas para su tipo de dato, pues como ya se dijo, se los tratará como si fueran variables independientes. Por lo tanto se le puede:
- Asignar un valor (←)

- Ej.: R(1) ← R(1) + 4
  - BAR(1,1) ← 5

- Ingresar un valor (leer)

- Ej.: leer (R(1)
  - leer (BAR(1,1))

- Exhibir su valor (escribir)

- Ej.: escribir (R(1)
  - escribir (BAR(1,1))

- Usar su valor como parte de cualquier expresión

- Ej.: R(1) - 3 > 0
  - BAR(1,1) * 3,12

Cuando se intenta acceder a una posición fuera del rango del arreglo se generará un error. La forma de manifestar este error difiere entre un lenguaje y otro, produciendo distintos efectos (en algunos casos se aborta el programa, en otros se presentan alterados los valores almacenados, etc.).
En este libro nos ajustaremos estrictamente a:

- gestionar los arreglos dentro de los límites fijados en su declaración.

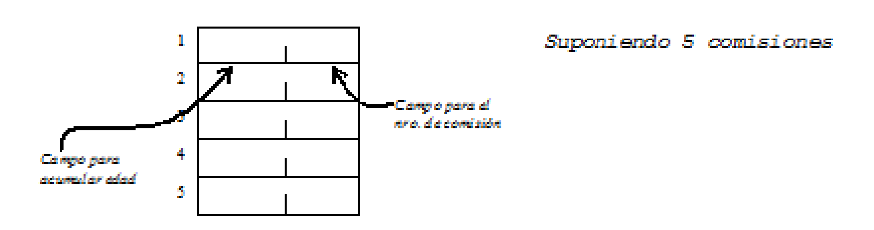
- implementar el acceso independiente a cada uno de sus elementos (si bien hay lenguajes que permiten acceder a todo el bloque de componentes sin distinguirlos).
- considerar que el arreglo NO quedará inicializado al ser declarado.

Ventaja del acceso directo a un elemento

La posibilidad de acceder directamente a cada componente del arreglo por medio del subíndice es lo que hace que el arreglo sea un recurso, particularmente apropiado para implementar datos compuestos en los que alguno de sus miembros, por los valores que pueda asumir, permite ser usado para hacer referencia a una componente.
Por ejemplo: Si, de la lectura del enunciado de una situación problemática, se llegara a la conclusión que:

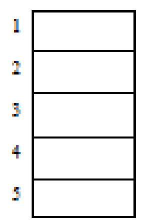
- Los datos a procesar son: *número de comisión y edad de cada alumno que pertenece a ella*.
- La forma de presentación de estos datos es: *pares de números enteros sin orden, siendo que los posibles valores del número de comisión son números consecutivos desde 1 en adelante hasta un valor determinado fijado en el enunciado*.
- Para obtener el resultado que se requiere es necesario: *calcular y mantener en memoria la suma de las edades de cada comisión*
- Para implementar el proceso de solución el recurso a usar es: *un arreglo para acumular la edad de cada alumno de cada comisión*

Podría pensarse en dos posibles formas de arreglo:
1°forma:
Un arreglo unidimensional de registros. El rango será = cantidad de comisiones (que coincide con el último número de comisión) y cada registro tendrá 2 campos: uno para el número de comisión y el otro para acumular las edades.



2°forma:
Un arreglo unidimensional de enteros: El rango será también la cantidad de comisiones y cada componente se usará para acumular las edades, usándose el número de comisión para referenciar al elemento correspondiente.

La 1°forma no sólo requerirá más espacio sino que exigirá más complejidad en el desarrollo del algoritmo en sí. Cada vez que se deba acumular una nueva edad habrá que “buscar entre todas las componentes del arreglo” la coincidencia entre el número de comisión de la edad a acumular y los existentes en el campo del arreglo reservado para él.



6.3.4 Ejemplos integradores de conceptos referidos a arreglos

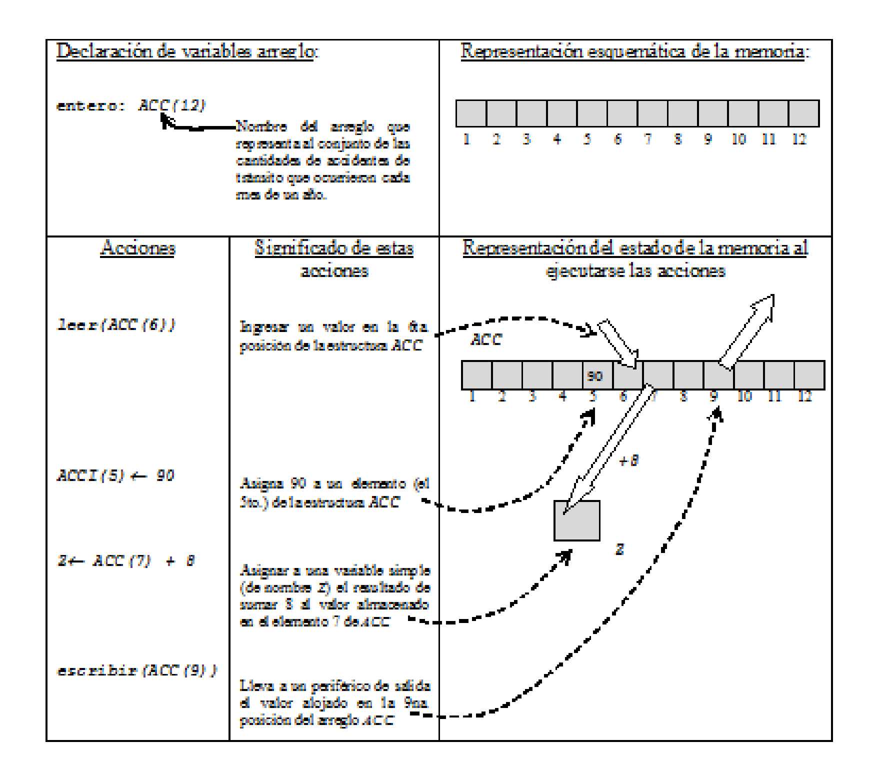
Para ejemplificar el uso de arreglo tomamos como punto de partida que, como consecuencia del análisis del enunciado del problema, se ha detectado un dato compuesto a procesar. Desde esta situación, realizamos:

1. un análisis de las características del dato, para resolver o no su implementación con un arreglo.
2. una descripción de las acciones que podrían estar en el algoritmo y junto a ellas la representación de lo que ocasionan en la memoria.

Ejemplo 1)
Dato que se necesita procesar: Cantidad de accidentes de tránsito registrados cada mes de un determinado año¿Qué tipo de dato usar?:
Evidentemente se trata de 12 números enteros. Si del análisis del enunciado en el que está mencionado este dato, surge la conveniencia de:

- trabajar este conjunto de valores como un todo
- mantenerlos disponibles en memoria para operar con ellos en distintos momentos
- tener la posibilidad de acceder a cada uno en forma individual

Se elegirá organizarlos como un arreglo unidimensional de enteros

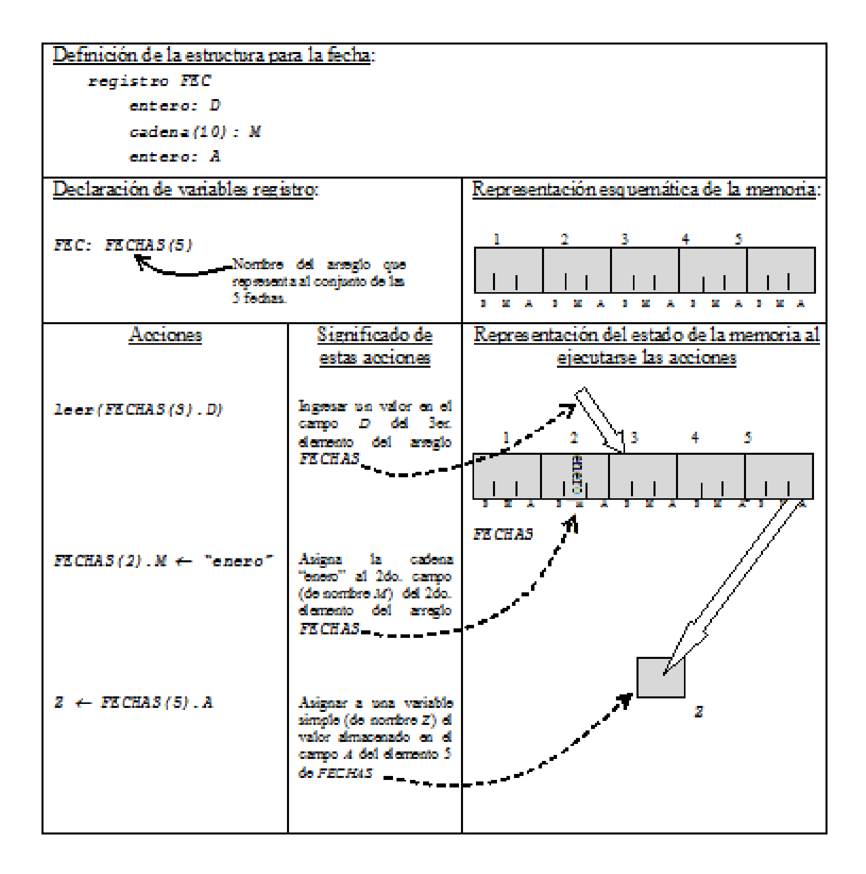


Ejemplo 2)
Dato que se necesita procesar: Un conjunto de 5 fechas expresadas en el formato: DDMESAAAA (nro. del día, nombre del mes, nro. del año).

¿Qué tipo de dato usar?:
Evidentemente se trata de 5 estructuras (o 5 variables compuestas), porque el valor correspondiente a una fecha no es de un tipo de dato estándar; sí podemos verla como un dato compuesto por 2 números enteros (día y año) y una cadena (nombre del mes).
Si del análisis del enunciado en el que está mencionado este dato, surge la conveniencia de:

- trabajar este conjunto de 5 valores como un todo
- mantenerlos disponibles en memoria, para operar con ellos en distintos momentos
- tener la posibilidad de acceder a cada uno en forma individual

Se elegirá organizarlos como un ***arreglo unidimensional de estructuras*** formadas por 2 enteros y 1 cadena.



6.3.5 Algoritmos de aplicación

En la parte inicial del capitulo, en **Ejemplos de usos de estructuras de datos**, se analizó el enunciado de distintas situaciones problemáticas, algunas de las cuales requieren el uso de arreglos para ser solucionadas. A continuación se presenta, a partir del análisis realizado a esos problemas, un algoritmo posible.
De un conjunto de números enteros positivos de dos cifras, se quiere determinar la cantidad de

veces que aparece cada número, en el conjunto.
En el algoritmo se usarán los siguientes identificadores de variables:
N: número entero de 2 cifras. Se elige el valor 0 como condición de fin
CN: arreglo donde se contarán los números ingresados
L y Z: para controlar la repetición fija

```
Algoritmo NUMEROS
 VARIABLES
   Enteros L, N, Z, CN(90)

  Inicio
   Repetir Para l←1, 90
          CN(L)←0
   Fin_para
   Leer (N)
   Repetir mientras (N <> 0)   !Se elige el valor 0 como condición de fin
     CN(N - 9)← CN(N - 9)+ 1
     Leer (N)
   Fin_mientras
   Repetir Para Z ←1, 90
        Si (CN(Z) <> 0) entonces
                         Mostrar (CN(Z))
        Fin si
   Fin para
  Fin
```

Se dispone de la edad y el nombre de 7 personas. Mostrar el nombre de los que tienen más edad que el promedio del grupo.

En el algoritmo se usarán los siguientes identificadores de variables:
pr_ed: edad promedio de las personas
sed: sumatoria de las edades de las personas (para poder calcular el promedio)
p: arreglo donde se mantiene la edad y el nombre de cada persona
L y N: para controlar la repetición fija
ed: edad de una determinada persona
nom: nombre de una determinada persona

*Los depósitos que se realizan en una cuenta bancaria, a lo largo de un año, se han ido registrando manualmente en distintas planillas. En ellas se tiene: número del mes del depósito, importe. En un mismo mes se pueden haber realizado uno, varios o ningún depósito. Esta información, que no está clasificada, debe ser procesada con la finalidad de conocerse el importe total depositado cada mes de ese año.*Este enunciado tiene las mismas características de los ejemplos planteados en **Acceso a un elemento del arreglo** y **Ventaja del acceso directo a un elemento**, donde se mostró dos formas distintas de organizar los datos en arreglos para un mismo contexto problemático.Al analizar este problema puede observarse que se plantea una situación similar a la allí descrita. A partir de ello se presentan a continuación, dos posibles algoritmos de solución. Si bien los dos algoritmos resuelven el problema, es evidente que una forma resulta más simple que la otra.Los datos a procesar son: número de mes e importe depositado.
Forma de presentación de estos datos: pares de números (mes – importe) sin orden; siendo los posibles valores del mes, cualquier número entero de 1 a 12.Para obtener el resultado que se pide es necesario: calcular y mantener en memoria la suma de los importes depositados cada mes.Recurso a usar: un arreglo para acumular los importes de cada depósito de cada mes.

```
Algoritmo EDADES
 TIPO
   Registro pers
                Entero ed
                Cadena(15) nom
VARIABLES
   Enteros L, N, Z, sed
   Real pr ed
   pers p(7)

 Inicio
   Repetir Para N←1, 7
          Leer (p(N).ed, p(N).nom)
   Fin para
   sed←0
   Repetir Para L←1, 7
     sed←sed + p(L).ed
   Fin para
   Pr_ed←sed/7
   Repetir Para N←1, 7
        Si (p(N).ed > Pr_ed) entonces
                 Mostrar (p(N).nom)
        Fin si
   Fin para
 Fin
```

1er. algoritmo

Se usarán los siguientes identificadores de variables:
MES: número de mes. Se elige el valor 0 como condición de fin
IM: importe del depósito
DEP: arreglo donde cada elemento tiene un campo para el número de mes y otro para acumular los importes del mes correspondienteL y Z: para controlar la repetición fijaM: permite referenciar en el arreglo la posición del registro correspondiente al mes a procesar.

```
Algoritmo DEPOSITOS
 TIPO
   Registro importe
              Entero n mes
              Real tot
 VARIABLES
   Enteros L, Z, M, MES
   Real IM
   importe DEP(12)

  Inicio
   Repetir Para L←1, 12
          DEP(L).n mes←L
          DEP(L).tot←0
   Fin_para
   Leer (MES)
   Repetir mientras (MES <> 0)
    !Se busca en el arreglo, a través del campo n_mes,la ubicación de MES
     M←1
     Repetir mientras (MES <> DEP(M).n_mes)
          M←1 + M
     Fin mientras
    !Finalizó la búsqueda
     Leer (IM)
     DEP(M).tot←DEP(M).tot + IM
     Leer (MES)
   Fin_mientras
   Repetir Para Z←1, 12
              Mostrar (DEP(Z).n_mes, DEP(Z).tot)
   Fin para
  Fin
```

2do algoritmo

Se usarán los siguientes identificadores de variables:
MES: número de mes. Se elige el valor 0 como condición de fin
IM: importe del depósito
DEP: arreglo donde se acumulan los importes de cada mes
L y Z: para controlar la repetición fija

```
Algoritmo DEPOSITOS
VARIABLES
   Enteros L, Z, MES
   Real IM, DEP(12)

  Inicio
   Repetir Para L←—1, 12
          DEP(L)←—0
   Fin_para
   Leer  (MES)
   Repetir mientras (MES <> 0)
     Leer  (IM)
     DEP(MES)←— DEP(MES)+ IM
     Leer  (MES)
   Fin mientras
   Repetir Para Z←—1, 12
             Mostrar  (Z, DEP(Z))
   Fin_para
  Fin
```

Operaciones con arreglos

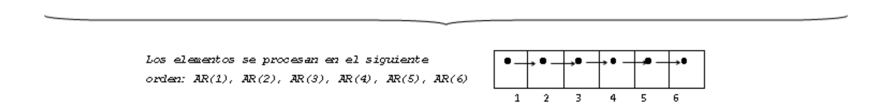
RECORRIDO o ACCESO SECUENCIAL:
Consiste en pasar por los sucesivos elementos del arreglo en forma ordenada hasta que no existan más posiciones disponibles: AR(1), AR(2), AR(3), ...
Esta operación puede realizarse con distintas finalidades:
- para asignarle algún valor a cada elemento
  Ejemplos:
    AR(2) ← “juan”
    leer (AR(2)
    TOR(7) ← TOR(7)* 32.5 + EQ

- para usar el valor contenido en cada elemento
  Ejemplos:
    escribir ZE(12)
    M ← TOR(3)

A continuación se presentan fragmentos de algoritmos que implementa un recorrido sobre un arreglo en memoria como así también su representación (en los gráficos el sentido del recorrido se muestra mediante una flecha →):Evidentemente para hacer el recorrido del arreglo lo que tiene que ir cambiando es la referencia de la posición, es decir el valor del índice. La forma como se va modificando la referencia origina distintos recorridos del arreglo.

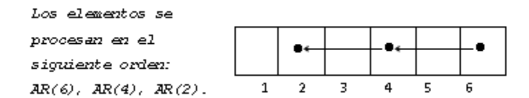
Arreglo unidimensional: sólo es posible un recorrido lineal.
Suponiendo declarado un arreglo de 6 enteros:
Recorrido completo (desde el 1er. al último elemento):

Usando una repetición fija:
Usando una repetición condicionada
Repetir Para ind ← 1, 6, 1 n ← 1
leer (AR(ind)) Repetir Mientras (n < 6)
fin_para AR(n) ← 3
n ← n + 1
El valor del índice lo maneja la variable *Fin_mientras*
de control de la repetición fija (ind).
En el caso de la repetición condicionada, el valor del índice lo maneja una variable especialmente definida, cuyo valor condiciona la repetición (n).

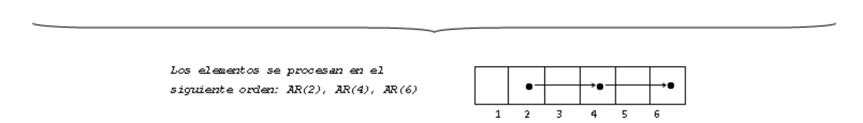


Recorrido parcial, a través de algunas posiciones:

1) Posiciones pares, desde el último al primero
Repetir Para L ← 6, 1, -2
Z(L) ← 0
Fin_para

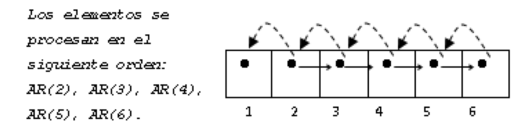


2) Posiciones pares, desde el primero al último
I ← 2 I ← 0
Repetir Mientras (I < = 6) Repetir Mientras (I < 6)
Escribir (AR(I) I ← I + 2
I ← I + 2 Escribir (AR(I)
Fin_mientras Fin_mientras



3) Posiciones adyacentes, desde la 2da. en adelante para desplazar los valores contenidos (los dos últimos elementos quedan con igual valor)
Repetir Para M ← 2, 6
AR(M - 1) ← AR(M)
Fin_para

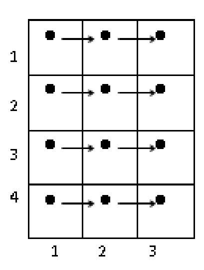
Arreglo bidimensional: es posible recorrerlo en dos sentidos.
- Por filas: cada elemento referenciado pertenece a una misma fila, por lo tanto para recorrerla deberá ir cambiando la referencia a la columna.
- Por columna: cada elemento referenciado pertenece a una misma columna, por lo tanto para



recorrerla deberá ir cambiando la referencia a la fila.

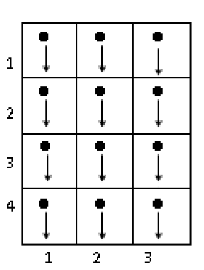
Suponiendo declarado un arreglo de 4 filas y 3 columnas de enteros:
Recorrido por fila completo:
Repetir Para L ← 1, 4, 1
  Repetir Para N ← 1, 3, 1
    MT(L,N) ← 0
  Fin_para
Fin_para

Los elementos se procesan en el siguiente orden: MT(1, 1), MT(1,2), MT(1,3), MT(2, 1), MT(2,2), MT(2,3), MT(3, 1), MT(3,2), MT(3,3), MT(4, 1), MT(4,2), MT(4,3)



Recorrido por columna completo:
Repetir Para L ← 1, 3, 1
  Repetir Para N ← 1, 4, 1
    MT(N,L)← 0
  Fin_para
Fin_para

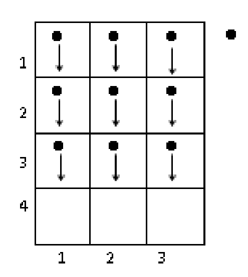
Los elementos se procesan en el siguiente orden: MT(1, 1), MT(2, 1), MT(3, 1), MT(4, 1), MT(1,2), MT(2,2), MT(3,2), MT(4,2), MT(1,3), MT(2,3), MT(3,3), MT(4,3)



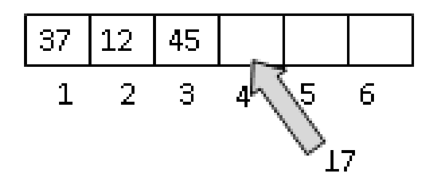
Recorrido que genera un ERROR DURANTE EL PROCESO:

Repetir Para L ← 1, 4, 1
  Repetir Para N ← 1, 3, 1
    MT(N,L) ← 0
  Fin_para
Fin_para
ACTUALIZACION:
Consiste en agregar, insertar o eliminar valores como si se tratara de una estructura extensible o

Los elementos se procesan en el siguiente orden: MT(1,1), MT(2,1), MT(3,1), MT(1,2), MT(2,2), MT(3,2), MT(1,3), MT(2,3), MT(3,3). Luego el índice de columna toma el valor **4** intentando procesar al **MT(1,4)** que NO EXISTE, abortándose el proceso. Por otro lado quedó una fila sin tratar.



reducible.
Siendo el arreglo una estructura estática, sólo mediante un trabajo algorítmico, podrá lograrse una “simulación” de estos procesos:- Agregar un valor a un elemento que estuviera “vacío”, es decir, que no contuviera un valor válido para el conjunto que se procesa.Por ejemplo se requiere agregar el valor 17 al 4to. elemento.



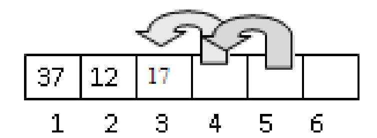
Problema de aplicación:
Suponemos organizado en memoria, un arreglo de 50 lugares para caracteres al que se le han cargado algunas letras mayúsculas (mucho menos de 50) en lugares sucesivos desde el primero en adelante.
*A partir de esto, agregar un *, en la posición siguiente a la de la última letra.*

```
Algoritmo AGREGADO
 VARIABLES
   Enteros I, Z
   Carácter C(50)
  Inicio
   !a continuación se inicializa el arreglo en un valor cualquiera
   Repetir Para I← 1, 50
          C(I) ← ' '
   Fin para

   !a continuación tiene que desarrollarse el proceso de
   !almacenamiento de menos de 50 caracteres en el arreglo
    - - - -
   !fin del proceso

   !a continuación se determina la última posición usada y se almacena el *
   Z ←1
   Repetir Mientras (C(Z)<> ' ')
        Z ←Z + 1
   Fin mientras
    C(Z)← '*'
  Fin
```

- Eliminar un valor. Esto se podría lograr desplazando hacia su posición a aquellos valores que están en posiciones posteriores a él (por ejemplo se elimina el valor 45 del 3er. elemento, copiando en ese lugar al valor que está en el 4to. elemento, y en éste, el que está en el 5to.)



El fragmento de algoritmo correspondiente a este desplazamiento, en un arreglo de nombre M, podría ser:

```
Repetir Para L ← P_elim + 1, P_ult
    M(L-1) ← M(L)
Fin_para
```

donde:

P_elim corresponde a la posición del arreglo que contiene un valor a eliminar

P_ult corresponde a la última posición del arreglo que contiene un valor válido

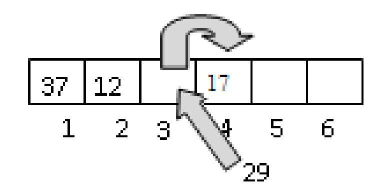
Problema de aplicación:

```
Algoritmo ELIMINACION
 VARIABLES
   Enteros K, Z, M, I, EN
   Carácter L(26)
  Inicio
   !a continuación se inicializa el arreglo en un valor cualquiera
   Repetir Para M ← 1, 26
          L(I) ← ' '
   Fin para
   !a continuación se guardan las letras del conjunto
   Repetir Para K ← 1, 15
          Leer(L(K))
   Fin para

   !a continuación se busca la R
   EN ← 0
   Z ← 1
   Repetir Mientras ( Z <= 26 y EN = 0)
        Si  (L(Z) = 'R') entonces
                          EN ← 1
                        sino
                          Z ← Z + 1
        Fin si
   Fin mientras
   !a continuación se elimina la R, si es que fue encontrada
   Si  (L(Z) = 'R') entonces
                     Repetir Para I ← Z+1, 26
                         L(I-1) ← L(I)
                     Fin_para
   Fin si
 Fin
```

- Insertar un valor. Esto se podría lograr desplazando hacia posiciones "vacías", a aquellos valores que están en posiciones posteriores a la de inserción (por ejemplo el valor 29 se puede insertar en el 3er. elemento, previa copia, del valor que estaba allí, en el 4to. elemento)

El fragmento de algoritmo correspondiente a este desplazamiento, en un arreglo de nombre M,



podría ser:
Repetir Para L ← P_ult, P_ins, -1
M(L+1) ← M(L)
Fin_para

donde:
P_ult corresponde a la última posición del arreglo que contiene un valor válido.
P_ins corresponde a la posición donde se quiere ubicar al nuevo valor

Problema de aplicación:
Ingresar a memoria un conjunto de 20 números enteros, ubicándolos en un arreglo de 150 elementos desde la 1º a la 20º posición del mismo.
A partir de esto, insertar en la posición 15º del arreglo el valor 999, tal que no se pierda ningún valor contenido en el arreglo ni se altere su secuencia de almacenamiento, quedando así 21 valores en memoria.

```
Algoritmo INSERSION
 VARIABLES
   Enteros K, Z, N(150)

  Inicio
   Repetir Para K←1, 20
          Leer(N(K))
   Fin_para
   Repetir Para Z← 20, 15, -1
        N(Z + 1)← N(Z)
   Fin para
    N(15) ← 999
  Fin
```

6.4 Actividad para los estudiantes

A continuación describimos situaciones problemáticas y luego planteamos algunas preguntas referidas a esos enunciados, que sugerimos las respondan:

Problema 1:
De cada uno de los alumnos de una institución educativa se tienen los siguientes pares de datos:
- Nº de identificación del curso al que asiste (número entero entre 1 y 7)
- Nota que obtuvo en el último examen parcial (número con parte decimal)

No se sabe la cantidad de alumnos por curso, sólo se puede asegurar que son 7 cursos. Estos pares de datos se presentan sin orden.

Determinar la nota promedio obtenida en cada uno de los cursos.

1. ¿Qué partes del texto de estos enunciados dan la pauta de que se trata de situaciones distintas?.
2. Proponer, para cada enunciado un posible lote de datos.
3. ¿En alguna de las dos situaciones problemáticas es necesario usar una estructura de datos para poder llevar a cabo el proceso que se requiere para arribar al resultado?. Si la respuesta fuera afirmativa especificar:
   - ¿Qué dice el enunciado, que te lleva a pensar de esa manera?.· ¿Qué tipo de estructura sería?.
   - ¿Para qué se la usaría?.
   - Esquematizar la estructura, indicando, de acuerdo con el enunciado, qué datos corresponderán a cada uno de sus componentes.
4. En cualquiera de los dos casos, los datos a procesar de cada alumno (N^{o}de identificación del curso al que asiste y nota que obtuvo en el último examen parcial) al ser ingresados:· ¿puede mantenérselos en memoria principal temporariamente, en una estructura de datos registro?· ¿puede mantenérselos en memoria principal temporariamente, en dos variables simples?

Problema 2:

De cada uno de los alumnos de una institución educativa se tienen los siguientes pares de datos:

- N^{o}de identificación del curso al que asiste (número entero)
- Nota que obtuvo en el último examen parcial (número con parte decimal)

No se sabe la cantidad de alumnos por curso, ni la cantidad de cursos. Estos pares de datos se presentan ordenados por número de curso.
Determinar, para cada uno de los cursos, la nota promedio obtenida por sus alumnos.

Responder las mismas preguntas que para el problema anterior.

Problema 3: Una inmobiliaria tiene información sobre 3 departamentos en alquiler, de cada inmueble se conoce:

- Código del inmueble (número entero)
- Superficie (número fraccionario)
- Precio del alquiler (número fraccionario)
- Disponibilidad (lógico)

Cuando un inquilino desea cancelar su contrato se debe modificar la disponibilidad del inmueble. Para ello se ingresa el código del departamento y automáticamente, se cambia el valor que tiene el estado de disponibilidad por su valor opuesto y se aumenta en un 12 % el precio del alquiler.

1. ¿Cuál es la información elemental que se quiere procesar?.
2. ¿Se podría usar una estructura de datos para guardar esta información elemental?. Si la respuesta fuera afirmativa especificar:
   - ¿Qué tipo de estructura sería?.
   - ¿Para qué se la usaría?.
   - Esquematizar la estructura, indicando, de acuerdo con el enunciado, qué datos corresponderán a cada uno de sus componentes

Problema 4:

En un club se tiene registrado de cada socio: edad y tipo de deporte que practica (1: rugby, 2: tenis, 3: voley). Determinar, para cada deporte, la edad promedio de sus practicantes como así también el nombre del deporte practicado por la mayor cantidad de personas. Se desconoce la

cantidad exacta de socios.

1. ¿Cuál es la información elemental que se quiere procesar?.
2. ¿Qué clase de variable (simple o estructurada) se usará para contener en memoria principal la edad promedio de cada deporte?
3. ¿Qué clase de variable (simple o estructurada) se usaría para contener en memoria principal el nombre de cada deporte?

Problema 5:

En un club se tiene registrado de cada socio: edad y tipo de deporte que practica (1: rugby, 2: tenis, 3: voley, 4: básquet, 5: fútbol, 6: natación, 7: atletismo, 8: patín, 8: jockey, 9: judo, 10: lucha, 11: water polo). Determinar, para cada deporte, la edad promedio de sus practicantes como así también el nombre del o los deportes practicados por la mayor cantidad de personas. Se desconoce la cantidad exacta de socios.

1. ¿Qué clase de variable (simple o estructurada) se usará para contener en memoria principal la edad promedio de cada deporte?
2. Si se conociera la cantidad exacta de socios, ¿se modificaría la manera de organizar los datos en memoria, para llegar a obtener la edad promedio de cada deporte?

Problema 6:

Una compañía de transporte posee un listado donde constan los valores de cada viaje que realiza, a lo largo de una semana laboral, cada una de sus 45 unidades. En esa planilla figuran los siguientes datos: importe del viaje y nro. de unidad que lo realizó (número entero entre 1 y 45).No se sabe la cantidad de registraciones que constan en la planilla, pues cada unidad pudo haber realizado más de un viaje. Los datos no están ordenados. Se desea obtener un listado donde conste, para cada número de unidad, el importe total recaudado en concepto de todos los viajes realizados por ella.

1. ¿Es necesario mantener en memoria principal una réplica de la planilla?. ¿Por qué?
2. ¿Cuántos importes aparecerán en el listado pedido?
3. ¿Qué importancia tiene, que se aclare en el enunciado, que el número de la unidad que realizó el viaje es un entero entre 1 y 45?

7 — Ordenamiento, Búsqueda e Intercalación

Fernando Guspí

Un gran descubrimiento resuelve un gran problema, pero en la solución de todo problema, hay cierto descubrimiento.

George Polya

7.0.1 Introducción

Presentar una lista o conjunto de elementos en un orden determinado, por ejemplo, alfabético o numérico, ascendente o descendente, es un problema que aparece con gran frecuencia en las aplicaciones de la Informática, así como la necesidad de buscar los elementos de un conjunto que cumplan determinada condición, donde el conjunto puede haber sido previamente ordenado o no. También es corriente tener que generar nuevos conjuntos a partir de otros ya ordenados, es así que los programas utilitarios generalmente contienen procedimientos para realizar estas tareas.En este capítulo estudiaremos los algoritmos que permiten resolver estos problemas y analizaremos su eficiencia, pues no en todos los casos los algoritmos llegan a los resultados esperados con igual facilidad o velocidad.

7.1 Ordenamiento

En la vida cotidiana, muchas veces nos disponemos a colocar objetos de una manera especial, siguiendo algún criterio, es decir, los ordenamos. Apilamos un conjunto de platos por tamaño, acomodamos varias carpetas por colores, clasificamos un conjunto de palabras por orden alfabético, acomodamos un mazo de cartas por palos o por números. En síntesis, clasificamos un conjunto finito de acuerdo con un criterio prefijado.Generalmente no nos detenemos a pensar cómo hacemos esto. Intuitivamente podemos decir que en una primera aproximación se determina cuál es el objeto que, de acuerdo con el criterio de ordenamiento fijado, debe ocupar la primera posición, y lo descartamos del conjunto. Luego se procede de igual forma, descartando los objetos ya ordenados.Un análisis más fino mostrará que el criterio de ordenar posición por posición requiere muchas comparaciones y desplazamientos de objetos, y en definitiva los algoritmos obtenidos no son muy eficientes para casos generales con grandes cantidades de datos. Entonces, también vamos a desarrollar otro algoritmo más eficiente, donde el ordenamiento se basa en dividir cada vez el conjunto en dos subconjuntos y ordenarlos por separado.De esta manera, se pueden elaborar diferentes algoritmos de ordenamiento, algunos de los cuales presentaremos a continuación.A los efectos de describir los algoritmos trabajaremos sobre el problema de ordenar una estructura de datos sencilla: el arreglo unidimensional, a partir del

cual las técnicas pueden extenderse a estructuras más complicadas.Por razones de ahorro de memoria, estudiaremos el ordenamiento de un arreglo sobre sí mismo, es decir sin emplear arreglos auxiliares, y consideraremos sólo el ordenamiento creciente, ya que para el decreciente bastaría cambiar el sentido de las desigualdades.

7.1.1 Ordenamiento por selección

Dado un arreglo de n componentes, el *método de selección* para ordenarlo en forma ascendente, es el siguiente:

- encontrar el menor valor del arreglo (*seleccionar el valor mínimo del arreglo)*
- intercambiar el elemento encontrado con el primero del arreglo.
- repetir estas operaciones con los n-1 elementos restantes, seleccionando, el segundo elemento, así sucesivamente.

A continuación veremos cómo funciona el método a través de un ejemplo.

Supongamos que disponemos de un arreglo unidimensional de 8 componentes enteras

8 6 -1 2 0 3 -4 3

Determinamos la componente con valor mínimo -4 (podría eventualmente haber más de una, en dicho caso seleccionamos la primera encontrada), y la intercambiamos con la primera componente del arreglo original. Al intercambiar resulta:
-4 | 6 -1 2 0 3 8 3

La primera componente del arreglo ha quedado ordenada, en el sentido de que es menor o igual que todas las siguientes . Ahora se trata de reordenar las n-1 componentes que siguen a la primera. Empleamos el mismo criterio: intercambiar la componente a ordenar con el mínimo de las que le siguen. Es decir, para ordenar la segunda componente, el mínimo es -1 y resulta:
-4 -1 | 6 2 0 3 8 3

donde las dos primeras componentes están ordenadas.

Así seguimos hasta ordenar la componente n-1, pues al ser ésta menor o igual que las que le siguen, la n queda también ordenada.
-4 -1 0 | 2 6 3 8 3

-4 -1 0 2 | 6 3 8 3

-4 -1 0 2 3 | 6 8 3

-4 -1 0 2 3 3 | 8 6

-4 -1 0 2 3 3 6 | 8

Veamos la implementación de un algoritmo para realizar este proceso. Para ello, sea un arreglo de elementos reales a ordenar. Suponemos $n < 10000$ y declaramos un arreglo de tamaño fijo de 10000 componentes, de las cuales ordenamos las n primeras.

```
 AlgoritmoSeleccion
entero n, i, j ,lugar
real V[10000], min

inicio
    Leer (n) ! cantidad de elementos, suponemos 0<n<=10000
    Repetir Para i ← 1, n
        Leer (V[i])
    fin para

    ! Comienza el ordenamiento
    RepetirPara i ← 1, n-1 ! i es la componente a ordenar
        min ← V[i]
        lugar ← i
        RepetirPara j ← i+1, n
            Si (V[j] < min) entonces
            min ← V[j] ! cambiar el mínimo
            lugar ← j ! posición del mínimo
            fin si
        fin para
        V[lugar] ← V[i] ! intercambiar
        V[i] ← min
    fin para

    Repetir Para i ← 1, n
        Escribir (V[i])
    fin para
fin
```

¿Qué sucede si n = 1? Simplemente el "Repetir Para" desde 1 hasta 0 no se ejecuta, y el arreglo de 1 elemento queda como está.Observar que en caso de haber dos o más componentes de igual valor mínimo, la que se selecciona es la que se encontró primero, pues la acción "si V[j] < min" implica que no se ejecuta con valores iguales.

7.1.2 Ordenamiento por intercambio o método de la burbuja

Este método comienza comparando cada elemento del arreglo con su siguiente, e intercambiándolos si es necesario para que queden ordenados.

Considerando el arreglo desordenado:

8 6 -1 2 0 3 -4 3

si queremos ordenarlo de menor a mayor, comparamos el primer elemento (8) con el segundo (6), y vemos que hay que intercambiarlos, obteniendo:

6 8 -1 2 0 3 -4 3

Luego hacemos lo mismo con el segundo (8) y el tercero (-1):

6 -1 8 2 0 3 -4 3

y así sucesivamente hasta comparar el elemento n-1 con el n-ésimo, obteniendo

6 -1 2 8 0 3 -4 3

6 -1 2 0 8 3 -4 3

6 -1 2 0 3 8 -4 3

6 -1 2 0 3 -4 8 3

6 -1 2 0 3 -4 3 | 8

Al final de esta primera pasada vemos que el mayor valor ha quedado en el último lugar, ha subido como si fuera una burbuja en un líquido, desplazando a los elementos menores. Es decir, la *n-ésima* componente del arreglo está ordenada, en el sentido de que es mayor o igual que todas las que la preceden. Entonces repetimos el proceso con las componentes anteriores, para que quede ordenada la n-1.

-1 6 2 0 3 -4 3 | 8

-1 2 6 0 3 -4 3 | 8

-1 2 0 6 3 -4 3 | 8

-1 2 0 3 6 -4 3 | 8

-1 2 0 3 -4 6 3 | 8

-1 2 0 3 -4 3 | 6 8

De esta manera podemos seguir intercambiando valores hasta tener ordenada la segunda componente, que será mayor o igual que todas las que la preceden, o sea la primera, y con esto finaliza el ordenamiento.Veamos la implementación de un algoritmo para realizar este proceso.

```
AlgoritmoBurbuja
entero n, i, j
real V[10000], aux
inicio
    Leer (n)
    Repetir Para i ← 1, n !suponemos 0<n<=10000
        leer(V[i])
    fin para
    ! Comienza el ordenamiento
    Repetir Para i ← n, 2, i -1
        ! i es la componente que quedará ordenada
        RepetirPara j ← 1, i-1
```

```
            Si (V[j] > V[j+1]) entonces
                aux ← V[j]
                V[j] ← V[j+1] ! intercambiar
                V[j+1] ← aux
            fin si
        fin para
    fin para
    Repetir Para i ← 1, n
        Escribir (V[i])
    fin para
fin
```

7.1.3 Ordenamiento por inserción

Este método, que recuerda la forma en que se ordena un mazo de naipes, comienza considerando que la primera componente del arreglo ya está ordenada, y toma la siguiente para determinar donde *insertarla* en la parte ordenada del arreglo (la izquierda). En ese momento se presentan tres posibilidades:

1. Si la que se está intentando *insertar* es menor que la primera, se la coloca en primer lugar, y se desplazan las restantes hacia la derecha.
2. Si la que se está intentando *insertar* es mayor que la última de la parte ordenada, queda en su misma posición, conformando la última del subconjunto ordenado.
3. Si la que se está intentando *insertar* no cumple ninguna de las dos condiciones anteriores, se la debe *colocar en un lugar intermedio*, desplazando las mayores a ella hacia la derecha.

Por ejemplo, para ordenar en forma ascendente el arreglo:

8 6 -1 2 0 3 -4 3

es decir, queda

6 8 | -1 2 0 3 -4 3

donde las dos primeras componentes aparecen en orden creciente.Ahora se considera la tercera componente, y recorriendo las componentes de la izquierda, se la "inserta" delante de la primera componente que sea menor, o bien al principio. En el ejemplo habría que insertar el –1 al principio, y desplazar el 6 y el 8.

Así:
-1 6 8 | 2 0 3 -4 3

las tres primeras componentes quedan en orden creciente.
Al tomar la cuarta componente, 2 , y recorrer el arreglo hacia la izquierda, vemos que $-1 < 2$,

luego insertamos 2 delante de 6 y desplazamos 6 y 8.

obteniendo:

-1 2 6 8 | 0 3 -4 3

Como observación: si la componente considerada es ya mayor que la anterior (p.ej: si en el paso anterior fuera 12 en vez de 2) simplemente se la deja como está, y se pasa a considerar la componente que sigue (el 0 en este caso).

-1 6 8 12 0 3 -4 3

Volviendo al arreglo original, ya nos damos cuenta, entonces, cómo serán los pasos posteriores, insertando las componentes restantes hasta la n

-1 0 2 6 8 | 3 -4 3

-1 0 2 3 6 8 | -4 3

-4 -1 0 2 3 6 8 | 3

-4 -1 0 2 3 3 6 8

Veamos la implementación de un algoritmo para realizar este proceso.

```
 Algoritmo Inserción
    entero n, i, j
    real V[10000], aux
    logico cond
inicio
    Leer (n) !suponemos 0<n<=10000
    Repetir Para i ← 1, n
        leer(V[i])
    finpara
    ! Comienza el ordenamiento
    Repetir Para i ← 2, n ! V[i] componente a insertar
        x ← V[i] ! almacenarla para no perderla
        j ← i – 1 ! posición componente anterior
        cond ← verdadero
        ! se puede comparar para insertar
        Repetir Mientras (cond y j>0)
            si x<V[j] entonces
                V[j+1] ← V[j] ! desplazar un lugar
                j ← j – 1
            sino
                cond ← falso
```

```
            finsi
        fin mientras
        V[j+1] ← x ! almacenar x en el lugar libre
    finpara
    Repetir Para i ← 1, n
        Escribir (V[i])
    finpara
fin
```

7.1.4 Cantidad de comparaciones efectuadas

Hasta aquí hemos propuesto algoritmos que efectivamente resuelven el problema de ordenamiento, es decir parten de un arreglo desordenado, y tomando cada uno por su camino, finalizan con el arreglo ordenado. Sin embargo, si la cantidad de datos a ordenar es grande, no todos los algoritmos trabajan con igual eficiencia. Un análisis de las comparaciones y operaciones efectuadas, nos detallará estas características.En los tres algoritmos que vimos, hay dos estructuras "repetir para" o "repetir mientras" anidadas, cada una de las cuales recorre una cantidad de elementos relacionada con *n*. Esto nos lleva a pensar, en un primer examen, que la cantidad de operaciones que realiza el algoritmo tiene que estar relacionada con n*n.Analicémoslo con más precisión en el método de la burbuja (en los otros el razonamiento es similar).
Hay n-1 componentes a ordenar. Para cada una de ellas se deben realizar entre 1 y n-1 comparaciones y eventuales intercambios. Tomando el promedio ((n-1)+1)/2 = n/2, la cantidad total de operaciones se puede estimar como:

(n-1) * n/2 =1/2*n*n - 1/2 * n= 1/2*(n*n – n)

Para n grande ($n \to \infty$), n es despreciable frente a n*n, y basta considerar sólo 1/2*n*n. Es decir, la cantidad de operaciones que hay que hacer para ordenar un arreglo de *n* componentes es proporcional a n*n. En otras palabras, la cantidad de comparaciones crece con el cuadrado de la cantidad de componentes del arreglo. Entonces, se dice, que el método de la burbuja es un algoritmo de orden n*n o cuadrático.Pensando que el tiempo de ejecución de un algoritmo es aproximadamente proporcional a la cantidad de operaciones o comparaciones que efectúa, si p. ej. para ordenar un arreglo de 10.000 componentes la máquina tarda 1 segundo, para ordenar uno de 100.000, cuya cantidad de componentes es 10 veces mayor, hay que esperar que demore un tiempo 10*10=100 veces mayor, o sea 100 segundos, que es 1 minuto con 40 segundos.Esta observación nos hace entrar en el terreno de la complejidad computacional. Hay algoritmos cuyo tiempo de ejecución crece mucho más que proporcionalmente a la cantidad de elementos a procesar. Por ejemplo, una tarea muy frecuente en problemas de cálculo técnico, es la resolución de sistemas lineales de orden n (sistemas de n ecuaciones con n incógnitas). Los algoritmos que resuelven estos sistemas por los métodos corrientes de eliminación, son del orden de n3. Pensemos que puede haber problemas que requieran algoritmos con un orden de exponente aún mayor, y nos daremos cuenta que tan importante como lograr mejoras en el hardware, p. ej. mediante computadoras más veloces, es obtener mejoras en el software, empleando o desarrollando algoritmos que reduzcan la complejidad de las tareas, o que puedan procesarlas en paralelo, etc.Concretamente, en el caso del ordenamiento de arreglos, no se ha inventado ningún algoritmo de orden n, pero sí del orden de *n log2 n* , el **Quick sort** u **Ordenamiento rápido**. Para ordenar un arreglo de 2 elevado a 7 = 128 componentes, el quick sort haría aproximadamente una cantidad de operaciones proporcional a *128 log2 128* = 128 x 7 = 896. Para ordenar uno de 2

elevado a 11 = 2048 = 16 x 2 elevado a 7 componentes, haría 2048 x 11 = 22.528 operaciones, o sea, unas 25 veces más. En cambio, el método de la burbuja haría 16 elevado a 2 = 256 veces más.

7.1.5 Recursividad

El algoritmo Quick sort puede describirse en forma compacta mediante un procedimiento que utiliza la recursividad. La recursividad es la posibilidad, común a muchos lenguajes, de que un subalgoritmo o subprograma se llame a sí mismo, y aquí la emplearemos para sintetizar en pocos pasos la operatoria del método Quick sort.

- Método del ordenamiento rápido (Quick sort)

Dado un arreglo desordenado

8 6 -1 2 0 3 -4 3

izq pivote der

Elegimos un elemento más o menos arbitrario llamado pivote, que puede ser una componente central. En este ejemplo tomamos el cero como pivote. La tarea básica del método es, mediante intercambios, realizar una partición del arreglo en dos subarreglos, de manera que en uno queden los elementos menores o iguales que el pivote, y en el otro los mayores o iguales (los elementos iguales al pivote, incluido éste, pueden quedar en cualquiera de los dos subarreglos).Para realizar la tarea, colocamos índices de posición izq y der en ambos extremos del arreglo. Avanzamos con izq hasta encontrar un elemento mayor o igual que el pivote y paramos. En el ejemplo, 8 ya es mayor que el pivote 0, luego no avanzamos más. Ahora avanzamos con der, hasta encontrar uno menor o igual que el pivote, en este caso -4.

8 6 -1 2 0 3 -4 3

izq pivote der

¿Se han cruzado izq y der? Todavía no, entonces intercambiamos ambos elementos, y desplazamos izq y der un lugar más hacia el centro

-4 6 -1 2 0 3 8 3

izq pivote der

Si ambos índices aún no se han cruzado, seguimos avanzando con el mismo criterio. Por izquierda quedamos en $6 > 0$ (6 es mayor que el pivote), mientras que por derecha, avanzamos hasta el pivote mismo.

-4 6 -1 2 0 3 8 3
izq der

Intercambiamos los elementos y desplazamos los índices, que siguen sin cruzarse.

-4 0 -1 2 6 3 8 3

izq der

En el paso siguiente, izq avanza una unidad hasta encontrar 2, mientras der pasa a señalar –1.Ahora sí están cruzados, y esto nos dice que la partición terminó, pues en efecto, los elementos desde el primero hasta der forman un subarreglo de elementos <= 0, mientras que desde izq hasta el último los elementos son >= 0.

-4 0 -1 2 6 3 8 3

der izq

¿Qué falta para ordenar completamente el arreglo? Ordenar por separado cada subarreglo, porque para ordenarlos ya no es necesario intercambiar elementos de un subarreglo con elementos del otro.Este ordenamiento se puede realizar mediante nuevas particiones en subarreglos, hasta llegar a arreglos de una única componente, que ya están ordenados, es decir en cada subarreglo, el procedimiento se llama recursivamente a sí mismo, para realizar nuevas particiones.

```
 Procedimiento qsort (E/S real: a [10000] ; E entero: n1, n2)
! n1 y n2 son los índices de las componentes entre las que se va a ordenar el arreglo
! inicialmente deben valer 1 y n respectivamente
     entero izq, der
     real pivot, aux
inicio
     izq ← n1
     der ← n2
     pivot ← a[(izq+der) div 2] ! medio aproximado
     Repetir
           Repetir Mientras (a[izq] < pivot)
                izq ← izq + 1
          fin mientras
          Repetir Mientras (a[der] > pivot) hacer
                der ← der - 1
          fin mientras
          Si (izq <= der) entonces
                aux ← a[izq]
                a[izq] ← a[der] ! intercambiar
                a[der] ← aux
                izq ← izq +1
                der ← der -1
          finsi
     hasta_(que izq >= der)
     Si (der > n1) entonces
          qsort (a, n1, der)
          ! el procedimiento se llama a sí mismo bajo cierta condición
     finsi
     Si (izq < n2) entonces
          qsort (a, izq, n2)
     finsi
FIN procedimiento
```

7.1.6 Cantidad de comparaciones y operaciones efectuadas

Cada partición de un arreglo de n componentes se realiza en a lo sumo n comparaciones e intercambios, dado que el arreglo se recorre una sola vez. En la segunda etapa, la partición de dos arreglos cuya suma de longitudes es n, requiere también una cantidad de operaciones del orden de n.¿Cuántas particiones son necesarias para ordenar el arreglo? Vamos a hacer un razonamiento muy simplificado, suponiendo que n es una potencia entera de 2, n=2 elevado k, p. ej. 2 elevado a 3 = 8 ó 2 elevado a 16 = 65536. Suponemos además que en cada partición el arreglo queda dividido en dos trozos de igual longitud.

(arreglo original) **. . . . .** 2 elevado a 3 = 8

1a. partición **. . . . | . . . .** 2 x 2 elevado a 2 = 8

2a. partición **. . | . . | . . | . .** 4 x 2 elevado a 1 = 8

3a. partición **. | . | . | . | . | . | . | .** 8 x 2 elevado a 0 = 8

(arreglo ordenado)

El esquema muestra que en cada partición, la longitud de los subarreglos se divide por 2, y esto lleva, considerando el conjunto de todos los subarreglos, a tener que efectuar k particiones, de n operaciones cada una para llegar a arreglos de una sola componente. Ahora bien, por definición de logaritmo, k = log2 n, por lo tanto, la cantidad de operaciones, donde cada una mueve n elementos, es del orden de **n log2 n.**

Ejemplos de ejecución

A modo de ilustración, se presentan los tiempos empleados por una PC corriente para ordenar 100000 enteros generados al azar, empleando programas codificados en Fortran basados en los algoritmos del presente capítulo.

Método de selección: 53 segundos
Método de la burbuja: 83 segundos
Método de inserción: 54 segundos
Método Quick sort: 1 segundo

Al margen de alguna diferencia entre los tres primeros algoritmos, que puede estar relacionada con los datos, se nota claramente la enorme diferencia de velocidad que se obtiene con el método quick sort, la cual, según se probó, es válida para un caso general. No obstante, si hay que ordenar pequeñas cantidades de datos, o en ciertos casos particulares en que los datos ya están parcialmente ordenados, los tres primeros algoritmos pueden todavía lograr alguna ventaja.

7.2 Métodos de Búsqueda

7.2.1 Búsqueda en un arreglo desordenado. Método de búsqueda

Se trata de determinar si entre las componentes de un arreglo cualquiera aparece o no determinado valor. En caso afirmativo dar la o las posiciones donde ello ocurre, y en caso contrario, informar que dicho valor buscado no se encuentra en el conjunto.Supongamos que queremos buscar el valor ***3*** en el siguiente arreglo:

8 6 -1 2 0 3 -4 3

Observamos que la 6^{a}y la 8^{a}componente tienen ese valor. Si se pide sólo la primera coincidencia, se puede detener el proceso en la 6^{a}posición, en caso contrario se deberá informar todas las posiciones donde se encuentra el valor buscado .En cambio, si se busca el valor 9, no hay ninguna componente con tal valor.Un algoritmo de búsqueda en un arreglo no necesariamente ordenado debe recorrerlo componente por componente. Por esa razón el método se denomina *secuencial*.A continuación veremos un ejemplo con un arreglo de enteros de hasta 10.000 componentes. Se quiere determinar si en él aparece o no un determinado valor, dando la posición de la primera ocurrencia.

```
Algoritmo Busqueda_secuencial
    entero n, i, valor, cont, X[10000]
inicio
    Leer (n) ! cantidad de elementos, suponemos 0<n<=10000
    Repetir Para i ← 1, n
        Leer (X[i]) ! leer cada componente
    finpara
    ! comienza la búsqueda
    Escribir (‘ingrese un valor a buscar’)
    Leer (valor)
    cont ← 1 ! contador de posiciones
    Repetir Mientras (cont < n y X[cont] <> valor)
        cont ← cont + 1
    finmientras
    Si (X[cont] = valor) entonces
        Escribir (’primera coincidencia de’, valor,’ está en posición’, cont)
    sino
        Escribir (valor,’no está en el conjunto’,)
    finsi
FIN
```

7.2.2 Cantidad de comparaciones efectuadas

La búsqueda secuencial en un arreglo desordenado es un algoritmo de orden n, pues deben recorrerse y compararse las n componentes del arreglo hasta localizar el valor buscado o hasta el final del mismo. Eventualmente, si la búsqueda finaliza en la primera coincidencia, la cantidad promedio de comparaciones sería n/2, que es también de orden n. Tal tipo de búsqueda no es eficiente cuando n es grande.Pensemos en buscar en una guía telefónica a qué abonado le corresponde un número determinado. Habrá que recorrerla leyendo número por número desde el principio, porque los números no están ordenados. En cambio, buscar qué número corresponde a

determinado abonado es mucho más sencillo, ya que los nombres de los abonados aparecen en orden. A continuación veremos un tipo de búsqueda más eficiente que aprovecha la ventaja de tener los datos ordenados, se trata de la búsqueda dicotómica.

7.2.3 Búsqueda en un arreglo ordenado. Método de búsqueda dicotómica

Sea un arreglo con *n* componentes *ordenado de menor a mayor*, en el cual queremos encontrar un valor dado. El método de *búsqueda dicotómica*, también denominado *búsqueda binaria*, consiste en localizar aproximadamente la posición media del arreglo y examinar el valor allí encontrado con respecto al que se está tratando de ubicar. Si este valor es mayor que el buscado, entonces, al estar ordenado, se descarta la segunda parte del conjunto y se procede a buscar en la primer parte del mismo. Se repite este proceso en la mitad correspondiente del conjunto, hasta que se encuentre el elemento buscado o bien, el intervalo de búsqueda haya quedado vacío. Si, por el contrario, el valor es menor al buscado, se prosigue con la segunda mitad del conjunto.El algoritmo correspondiente es el siguiente:

```
Algoritmo busq_dicot
    entero: arreglo X[1,10000]
    entero: n, i, min, max, medio, valor
inicio
    Leer (n)
    Repetir Para i ← 1, n
        Leer(x[i])
    finpara
    Leer (valor) !valor a buscar
    min ← 1
    max ← n
    medio ← (min + max) div 2
    ! comienza la búsqueda
    Repetir Mientras (X[medio] <> valor y min <= max)
        Si (valor < X[medio]) entonces
            max ← medio - 1
        sino
            min ← medio + 1
        finsi
        medio ← (min + max) div 2
    finmientras
    Si (X[medio] = valor) entonces
        Escribir (valor,'en posicion',medio)
    sino
        Escribir (valor,'no está')
    finsi
fin
```

Analicemos el algoritmo a partir de un arreglo ordenado de menor a mayor de 9 componentes enteras, buscando distintos datos.

-8 -5 -2 0 1 1 4 6 7

1) Búsqueda del número –5.

En la primera asignación resultan min $\leftarrow$ 1, max $\leftarrow$ 9, medio $\leftarrow$ 5, de donde x[medio] $\leftarrow$ 1, que es mayor que el dato –5. Por lo tanto, hay que seguir buscando hacia la izquierda, lo que significa trasladar max a centro – 1, o sea max $\leftarrow$ 4. Promediando con min queda medio $\leftarrow$ (1 + 4) div 2 = 2, y precisamente en la posición 2 se encuentra –5, por lo que la búsqueda termina aquí.

2) Búsqueda del número 23

La primera vez, siendo medio la posición 5, resulta 23 $>$1, y hay que buscar a la derecha, o sea hacermin $\leftarrow$ medio +1 = 6. Promediando, medio $\leftarrow$ (6 + 9) div 2 = 7, donde tampoco está el dato. Como 23 $>$ 7, hacemos min $\leftarrow$ 8, medio $\leftarrow$ (8 + 9) div 2 = 8, no está el dato, llevamos min a 9, y centro $\leftarrow$ (9 + 9) div 2 = 9. La posición 9 contiene el valor 7, que no es el dato 23. Como 23 $>$ 7, ahora queda medio $\leftarrow$ centro +1 = 10. Se han cruzado izq y der, y al quedar vacío el intervalo de búsqueda la condición establecida en el "Repetir mientras" hace que la búsqueda finalice sin éxito.

3) Búsqueda del número 7.

La respuesta del método a este caso deberá ser 9, es decir, la posición del número 7 en el arreglo.Realizando la misma metodología que antes, la posición media del arreglo es 5, medio=(min+max) div 2, donde X[medio] es 1, que es menor que el valor 7. Por lo tanto, hay que seguir buscando hacia la derecha, y como la posición 5 ya se examinó, el nuevo intervalo de búsqueda resulta [medio+1, max]= [6,9], donde min = medio+1.Calculando nuevamente el medio del nuevo intervalo, medio=(6 + 9) div 2 = 7, donde X[medio] es 4, que es menor que el valor 7. Por lo cual habrá que correr el intervalo de búsqueda hacia la derecha, y como la posición 7 ya se examinó, el nuevo intervalo de búsqueda resulta [medio+1, max]=[8,9], donde min = medio+1. Calculando nuevamente el medio del nuevo intervalo, medio=(8 + 9) div 2 = 8, donde X[medio] es 6, menor que el valor 7. Por lo cual habrá que correr el intervalo de búsqueda hacia la derecha, y como la posición 8 ya se examinó, el nuevo intervalo de búsqueda resulta [medio+1, max]=[9,9]. Calculando nuevamente el medio del nuevo intervalo, medio=(9 + 9) div 2 = 9. Precisamente en la posición 9 se encuentra el valor 7 buscado, por lo tanto la búsqueda termina aquí.

7.2.4 Cantidad de comparaciones efectuadas

Razonando en forma simplificada, como lo hicimos con el método Quick sort, pensemos que la cantidad de componentes del arreglo es n = 2 elevado k. Luego de la primera operación, o sea del primer cálculo y comparación en el centro, la longitud del arreglo donde hay que seguir buscando se redujo a la mitad, 2elevadok-1. Luego de la segunda, la longitud es 2 elevado k-2, y después de k operaciones, la longitud es 2 elevado 0 = 1, donde se efectúa todavía una comparación más (eventualmente el proceso podría haberse detenido antes, de coincidir el dato en alguno de los centros intermedios). En conclusión, la cantidad promedio de comparaciones es k + 1 = (log2 n) +1. Si $n >>1$, el 1 puede despreciarse, y se dice que el algoritmo de búsqueda dicotómica es del orden de **log2 n.**
Por ejemplo, buscar en forma secuencial un abonado en una guía de 100.000 teléfonos requiere efectuar entre 1 y 100.000 comparaciones, como promedio 50.000. En una búsqueda dicotómica, en cambio, esta cantidad se reduce a log2 100.000 $\approx$ 17 comparaciones.

7.3 Método de intercalación

El método de Intercalación, también conocido como Merge en inglés, consiste en intercalar los valores de dos arreglos ordenados de igual manera, tal que se forme un tercer arreglo igualmente ordenado con los elementos de los dos arreglos originales.Considerando dos arreglos ordenados en forma ascendente:

A = (1, 7, 18, 23)

B = (2, 5, 6, 9, 12, 16, 17)

se trata de componer un nuevo arreglo ordenado:

C = (1, 2, 5, 6, 7, 9, 12, 16, 17, 18, 23)

donde los elementos de C se han obtenido intercalando las de A y B.

Un criterio muy poco eficiente para resolver el problema sería construir un arreglo D colocando los elementos de B a continuación de los de A y luego reordenarlo partiendo de D = (1, 7, 18, 23,2, 5, 6, 9, 12, 16, 17)En esta forma se pierde la ventaja de que A y B ya hayan sido ordenados, y aumenta el tiempo de ejecución.

El método de Intercalación compara los primeros elementos de A y de B y coloca en C el menor de ellos (por estar ordenados en forma ascendente), descartando dicho elemento. Luego continúa con los restantes, hasta que alguno de los dos arreglos se haya traspasado completamente a C, quedando las últimas componentes del otro arreglo por pasar. En el caso del ejemplo, los valores del arreglo B serán pasados a C, quedando el 18 y el 23 del arreglo A por pasar.

```
 Algoritmo intercalar
    arreglo [1,10000] real: A, B
    arreglo [1,20000] real: C
    ! C debe contener las comp. de A y las de B
    entero: n, m, i, inda, indb
inicio
    Leer (n) ! cantidad de componentes de A
    Repetir Para i ← 1, n
        Leer (A[i])
    finpara
    Leer (m) ! cantidad de componentes de B
    Repetir Para i ← 1, m
        Leer (B[i])
    finpara
    inda ← 1
    indb ← 1 ! próxima componente en cada arreglo
    ! intercalación
    Repetir Para i ← 1, n + m ! llenar cada componente de C
        Si (inda <= n y indb <= m) entonces
            ! hay componentes disponibles
            Si (A[inda] < B[indb]) entonces
                C[i] ← A[inda]
```

```
                inda ← inda + 1
            sino
                C[i] ← B[indb]
                indb ← indb + 1
            finsi
        sino
            Si (inda > n) entonces !se agotó A
                C[i] ← B[indb]
                indb ← indb + 1
            sino
                Si (indb > m) entonces ! se agotó B
                    C[i] ← A[inda]
                    inda ← inda + 1
                finsi
            finsi
        finsi
        Escribir (C[i])
    finpara
fin
```

Se propone verificar paso a paso este algoritmo con los arreglos que se dieron como ejemplo.

7.3.1 Cantidad de comparaciones efectuadas

Puede notarse en el algoritmo que la cantidad de componentes del arreglo ordenado (n+m) se recorre una sola vez, y que el recorrido, cualquiera sea el caso, incluye sólo una comparación por componente. Luego, el orden del algoritmo es *n+m*, o llamando directamente *n* a este total de elementos, el orden es *n*. En cambio, ordenar el arreglo D propuesto al principio, aún empleando Quick sort, insume *n log2 n* operaciones, es decir *log2 n* veces más.

7.3.2 Ejemplos adicionales

Mejoramiento del método de la burbuja

Hemos visto que en términos generales, para ordenar un conjunto de n elementos arbitrariamente desordenados, el algoritmo de Intercambio o Burbuja, que debe realizar n-1 pasadas, es ineficiente. Sin embargo, si el arreglo de datos ya está parcialmente ordenado, puede ocurrir que quede totalmente ordenado antes de completar todas las pasadas. Si en una pasada el arreglo ya quedó ordenado, el proceso se puede interrumpir allí, ahorrando de esta manera el esfuerzo de las pasadas siguientes. La cuestión entonces es cómo saber si el arreglo quedó ordenado, y la respuesta es: si en una pasada completa no fue necesario intercambiar elementos, quiere decir que ya están en orden. Habrá que introducir entonces en el algoritmo un contador de intercambios, o bien una bandera que tome valor Verdadero si en una pasada no se intercambiaron elementos (arreglo ordenado) o Falso, en caso contrario. Escribimos el pseudocódigo resultante:

```
AlgoritmoBurbuja1
    entero n, i, j
    real V[10000], aux
    logico ordenado ! bandera para detectar si el arreglo está ordenado
```

```
inicio
    Leer (n)
    Repetir Para i ← 1, n
        leer(V[i])
    fin para
    ! Comienza el ordenamiento
    ordenado ← falso
    i ← n
    Repetir Mientras (i>=2 y no(ordenado)) ! cada pasada
        ordenado ← verdadero ! si no cambia es porque está ordenado
        Repetir Para j=1, i-1
            Si (V[j] > V[j+1]) entonces
                aux ← V[j]
                V[j] ← V[j+1] ! intercambiar
                V[j+1] ← aux
                ordenado ← falso ! al haber intercambio no está ordenado
            fin_si
        fin_para
        i ← i-1
    fin mientras
    Repetir Para i ← 1, n
        Escribir (V[i])
    fin para
fin
```

Observar que las pasadas, cuya cantidad no se conoce a priori, se controlan mediante una estructura Repetir mientras, a diferencia del caso original, donde al conocerse la cantidad, se utiliza una estructura Repetir Para.

Ordenamiento de las filas de una matriz de acuerdo a los valores en una columna

Consideremos una matriz o arreglo bidimensional de n filas y m columnas, p. ej.

7 8 -4
5 1 0
-2 6 -1
3 3 2

donde queremos reordenar sus filas, de tal manera que los valores de la primera columna queden en orden creciente. El resultado es
-2 6 -1
3 3 2
5 1 0
7 8 -4

Observar que esto no es lo mismo que reordenar sólo la primera columna, porque en ese caso, las demás columnas no se moverían, y quedarían igual que en la matriz original. Entonces, para resolver el problema aplicando algún método de ordenamiento, la idea es aplicar el método a la primera columna, pero cada vez que se realiza un intercambio, intercambiar además todos los elementos de la correspondiente fila. Como ilustración desarrollaremos el algoritmo empleando el método de selección o búsqueda del mínimo.

```
Algoritmo Orden_filas
    entero n, i, j ,lugar
    real A[1000,1000], min, aux
inicio
    Leer (n, m) ! dimensiones de la matriz
    Repetir Para i ← 1, n
        Repetir Para j ← 1,m
            Leer (A[i,j])
        finpara
    finpara
    ! Comienza el ordenamiento
    Repetir Para i ← 1, n-1 ! i es la fila a ordenar
        min ← A[i,1]
        lugar ← i
        Repetir Para j ← i+1, n
            Si (A[j,1] < min) entonces
                min ← A[j,1] ! cambiar el mínimo
                lugar ← j ! posición del mínimo
            finsi
        finpara
        Repetir Para j← 1, m ! intercambiar toda la fila
            aux ← A[i,j]
            A[i,j] ← A[lugar,j]
            A[lugar,j] ← aux
        finpara
    finpara
    Repetir Para i ← 1, n
        Repetir Para j ← 1, m
            Escribir (A[i,j])
        finpara
    finpara
Fin
```

8 — Estructuras de datos: archivos

Cristina I. Alarcón - Zenón Luna

Los analfabetos del siglo XXI no serán aquellos que no sepan leer y escribir, sino aquellos que no puedan aprender, desaprender y reaprender

Alvin Toffler (1979)

8.0.3 Introducción

Las organizaciones de datos que conocemos hasta el momento son los arreglos y los registros. En ambas estructuras u organizaciones los datos residen en memoria principal. Esto implica los siguientes inconvenientes:

- El volumen de datos dependerá del tamaño de la memoria principal.
- El tiempo de residencia de los datos en esta memoria estará supeditado al encendido del equipo y ejecución del programa.

Es evidente que para el manejo de grandes volúmenes de información no son apropiadas.Para poder superar estas dificultades se creó un tipo de estructura de datos que asegura un almacenamiento permanente de los mismos en memorias secundarias o masivas como ser discos, CD, DVD, BLU RAY, etc. En este tipo de estructura los grandes volúmenes de datos se encuentran fraccionados en unidades más pequeñas que pueden ser almacenadas en memoria principal y tratadas por un programa (Registros Lógicos). Estas estructuras se llaman archivos o ficheros.

8.0.4 Características de los archivos

- Un archivo siempre está en un soporte externo de almacenamiento.
- Existe independencia de los datos respecto a los programas.
- La información guardada en un archivo es permanente.
- Los archivos permiten una gran capacidad de almacenamiento.
- No tienen tamaño definido, el tamaño del mismo dependerá de la capacidad disponible en la memoria auxiliar donde se vaya a grabar.
- Un archivo o fichero está formado por una cantidad no determinada de elementos. Estos elementos son llamados generalmente registros lógicos.

Un registro lógico es un conjunto de información relacionado lógicamente, perteneciente a una entidad y que puede ser tratado como una unidad por un programa.Los campos o elementos de un registro lógico pueden ser variables simples o estructuradas.

Ejemplos:

Ejemplo 1: datos Personales

Apellido_ nombres
Tipo_nro_documento
Edad
Nacionalidad
Domicilio

Campos o elementos del registro lógico datos personales: Apellido_ nombres, Tipo_nro_documento, Edad, Nacionalidad, Domicilio

Ejemplo 2: datos días

nro_ dia
nro_ mes
temperatura_máxima
temperatura_ mínima

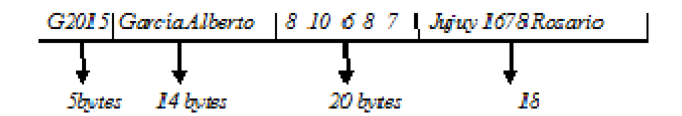
Campos o elementos del registro lógico datos días: nro_ día, nro_ mes, temperatura_ máxima, temperatura_mínima.

La transferencia de información entre el dispositivo de almacenamiento masivo y la memoria central se hace a través de una memoria intermedia o buffer que no es otra cosa que una partición de la memoria central del ordenador.El tamaño de esta memoria buffer es generalmente el tamaño del bloque o registro físico.

Registro físico o bloque: En el disco la información está escrita "físicamente", y la cantidad más pequeña de datos que puede transferirse en una operación de lectura x2F; escritura entre la memoria y el dispositivo periférico se denomina registro físico.

Ejemplos: un sector de un disco (512 Bytes), una línea de impresión (si el dispositivo es la impresora), etc.
Un registro físico puede contener uno o más registros lógicos, de acuerdo al tamaño del registro lógico.

Ejemplo:
Si tenemos un archivo llamado Alumnos.dat cuyo registro lógico **alum** es el siguiente:



Podemos, entonces, deducir su tamaño:
Tamaño de registro lógico alu1 =5 +14+20+18 =57 bytes
Tamaño del registro físico = 512 bytes

Coeficiente de transferencia o factor de bloqueo = 512 x2F; 57 = 8
Por lo tanto en la lectura de información desde el disco a la memoria se podrán “bajar” 8 registros de una sola transferencia. También decimos que el coeficiente de transferencia es 8.
La importancia de un factor de bloqueo mayor a 1, es que reduce la cantidad de operaciones de Ex2F;S entre la partición de la memoria RAM llamada buffer, donde se almacenarán los datos en forma transitoria de una operación de E ó S, y el dispositivo de almacenamiento masivo donde se encuentra almacenado el archivo de datos. Reducir el número de operaciones significará un menor tiempo en el procesamiento de la información contenida en el archivo.

Bases de datos
Los archivos pueden a su vez agruparse en estructuras de datos que permiten almacenar y procesar metódicamente grandes cantidades de información. Una **base de datos** es un conjunto de archivos con datos relacionados, organizados de acuerdo a un modelo, p. ej. el llamado modelo relacional. Dentro del modelo, se definen operaciones entre los archivos, que dan como resultado otros archivos. Hay lenguajes como SQL (structured query language), que efectúan directamente estas operaciones, ahorrando así gran cantidad de pasos de programación convencional.

Tipos de soportes
El soporte es el lugar físico donde se va a almacenar la información.

Estos soportes en general pueden ser:
- **Secuenciales** los registros están dispuestos uno a continuación del otro, para acceder al registro X, hay que “pasar” por los X -1 anteriores. Ejemplo: cintas magnéticas o líneas de la impresora.
- **Direccionales** por ejemplo los discos magnéticos, CD, DVD, y también en las memorias extraíbles, la información puede ser accedida a través de su dirección, o sea ir directamente a la posición de memoria donde está el registro que interesa, sin pasar por otros registros. Eventualmente, los soportes direccionales pueden también ser utilizados como soportes secuenciales.

Acceso
La forma de acceder a los registros de un archivo depende del tipo de soporte y de la forma en que fueron
almacenados los mismos.
- **Acceso secuencial**: implica el acceso desde el primer registro hasta el registro requerido. El soporte puede ser secuencial yx2F;o direccional.
- **Acceso directo**: esto implica acceder a un registro determinado. Este tipo de acceso s**ó**lo se da con soportes direccionales.

8.1 Organización de Archivos

Es la forma en que se disponen los registros que forman el archivo cuando se los crea.

a) Organización secuencial

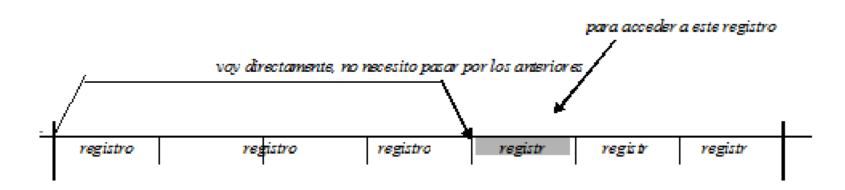
Puede emplearse con soportes secuenciales o direccionales. Los registros se estructuran o almacenan o graban uno a continuación del otro. El orden físico coincide con el orden lógico. En general el último registro contiene al final del mismo una marca de fin de archivo. Los registros pueden tener distinta longitud.EOF (END OF FILE) es una macro o función. Esta macro devuelve un valor lógico Verdadero cuando encuentra la marca de fin de archivo, marca que se encuentra a continuación del último registro. Esta marca no se visualiza y puede ser un

*, #, etc. Esta función se utiliza en la lectura de archivos secuenciales. En escritura, al cerrar el archivo, el procesador coloca la marca EOF. Reiteramos, en los archivos secuenciales, para acceder a un registro, se debe pasar por los anteriores, no se puede acceder directamente a un registro dado. Leer o escribir en un archivo secuencial significa siempre "en el registro siguiente" (o en el primero cuando se abre).

b) Organización directa

Sabemos que en este tipo de organización se accede al registro sin necesidad de pasar por los registros que le anteceden, ya que el procesador junto a su sistema operativo, los ubica en direcciones de disco o memoria que obtiene a través de un cálculo. Si todos los registros tienen el mismo tamaño y están almacenados consecutivamente, conociendo el tamaño de cada uno, le es simple al procesador localizar el registro número N, con independencia de los demás. Entonces se hace necesario informar al compilador cómo son los registros, pues ya dijimos que en un archivo directo son todos del mismo tipo. Esto se efectúa habitualmente al abrir el archivo. No es necesario, en cambio, especificar el modo de habilitación, porque los archivos directos se abren siempre como entrada-salida.

Si bien la sintaxis de las declaraciones y operaciones varía considerablemente de un lenguaje a otro, es esencial que en cada operación de lectura o escritura se indique específicamente el número relativo del registro en el que se va a leer o escribir. Aquí ya no hay "registro siguiente". El ejemplo que sigue busca dar una idea de cómo se trabaja con archivos directos, procurando no atarse a formas rígidas de sintaxis.



Sólo puede emplearse con soportes direccionales. El procesador debe conocer o calcular la dirección de cada registro a partir de su número relativo (1, 2, 3,..), para poder posicionarse directamente en él.

c) Organización indexada

Consideremos un archivo de organización directa cuyos registros, todos de la misma forma (p. ej. el tipo **alum** declarado anteriormente), contienen campos referidos a alumnos de una asignatura:

legajo	apeynom	notas			domicilio
G2015	GARCIA ALBERTO	8 10 6 8 7	JUJUY	1678	ROSARIO
P2436	PEREZ JUAN	7 3 8 5 6	ALSINA	380	PERGAMINO
R3118	RODRIGUEZ ANA	9 9 9 7 10	PASCO	1220	ROSARIO
L3223	LOPEZ JOSE	5 5 6 7 2	SALTA	927	PARANA

Cada registro está representado por una línea, y al procesar el archivo podemos acceder directamente a él, sin pasar por los anteriores, si conocemos su posición, número de orden o número relativo.

Llamamos **campo clave** a un campo que toma valores distintos en cada uno de los registros del archivo. En este caso, el legajo es seguro un campo clave, pues cada alumno aparece una sola

vez, y entonces legajodefine unívocamente al registro que lo contiene. El campo nombre podría eventualmente pensarse como clave, pero puede darse el caso de homónimos. Las notas y el domicilio no son claves, pues pueden repetirse en distintos registros.En las aplicaciones, puede ser importante acceder a un registro conociendo su clave, p. ej. 191;Qué nota obtuvo el alumno de legajo R3118?, pero a un archivo directo sólo podemos acceder por número relativo de registro: el 1, el 2, el 26. Si no sabemos en qué registro están los datos del legajo R3118, tendremos que acceder primero al registro 1, si el legajo no coincide ir al 2, etc. y esto no es más que una búsqueda secuencial, que es lenta.La organización indexada soluciona el problema agregando al archivo directo un archivo secuencial complementario llamado índice, cuyos registros contienen, al lado de cada valor de la clave, el número de registro en que ésta se encuentra en el archivo principal.

INDICE

clave no.	de registro
A1111	7
L3223	4
P2436	2
R3112	3
Z0325	1

Las claves habitualmente están ordenadas. Entonces, para procesar el registro cuya clave es R3112, vamos al índice y hacemos una búsqueda binaria o dicotómica, según el método que ya hemos desarrollado en el capítulo correspondiente. El resultado de la búsqueda, 3 en este caso, nos permite ahora sí acceder en forma directa al registro número 3 del archivo principal, sin tener que pasar por los anteriores. En la organización indexada, que es común en bases de datos, cada vez que se modifica el archivo principal, se actualiza automáticamente el índice.

a.1) Instrucciones para el manejo de archivos con Organización Secuencial

Orden abrir()

En nuestro pseudocódigo la orden Abrir significara: HABILITAR

La forma o sintaxis es:

abrir (nro de comunicación lógico, “Path o ruta nombre del archivo”, acción= “modo”)

Donde:

nro de comunicación lógico o canal de comunicación es un número ENTERO, que se asigna al archivo al habilitarlo. En cada operación posterior que deba realizase con ese archivo se lo identificará a través de ese número y no por medio de su nombre. Algunos lenguajes, en vez del nro, emplean un nombrede variable o alias para identificar al archivo. **“Path o ruta nombre del archivo”** es el lugar donde se encuentra o encontrará el archivo habilitado,seguido del nombre del mismo. **modo o forma de su habilitación**, puede asumir los valores:

Escritura, el archivo habilitado es para crearlo, se graba o escribe un archivo nuevo.

Lectura, el archivo habilitado es leído y ya debe estar creado.

Lectura-Escritura, permite abrir un archivo para lectura yx2F;o escritura. Existe el peligro

de sobrescribir registros ya grabados

Agregado, se escriben nuevos registros a continuación del último existente.

Orden leer()
Esta orden permite leer o ingresar valores "del registro siguiente" de un archivo ya creado, y su forma es:

leer (nro de comunicación lógico, lista de variables)

Orden escribir()
Esta orden permite escribir valores "en el registro siguiente", y su forma es:

escribir (nro de comunicación lógico, lista de variables)

Orden cerrar()
Por medio de esta orden se cierra todo tipo de comunicación con el archivo habilitado, identificado por el número de comunicación lógico o canal de comunicación. Además, en modo escritura, se coloca la marca de fin de archivo. Su forma es: **cerrar (nro de canal)**

Siempre que se habilita un archivo, sea para lectura, escritura o lectura-escritura, cuando se concluye la utilización del mismo debe cerrarse la comunicación con él, sino se corre el riesgo de perder toda la información del archivo.

a.2) Operaciones básicas de archivos con Organización Secuencial: Consulta y Actualización

Consulta: consiste en poder examinar uno o más registros de un archivo.

Actualización: comprende las siguientes tres operaciones:

A - **alta** significa agregar uno o más registros al archivo.

B - **baja** consiste en eliminar uno o más registros del archivo.

M - **modificación** consiste en modificar uno o más campos de un determinado registro.

No quiere decir que estas operaciones tengan que hacerse separadas, cada una en un programa o subprograma. Un mismo programa puede leer datos de un archivo, realizar operaciones, guardar resultados en otro, modificar o eliminar los registros que hagan falta, etc.Aquí presentamos ejemplos separados de altas, bajas y modificaciones, simplemente para ilustrar mejor cómo trabaja cada uno.

- ALTAS

Hacer un algoritmo que permita agregar (ALTAS) registros al archivo DATOS.DAT, cuyos registros tienen
los campos que vimos en el tipo alum.

```
Algoritmo Altas
Tipo registro alumno
    carácter(*5) legajo
        tipo registro apeynom
            carácter(*20) apellido
            carácter(*25) nombre
        fin
    entero notas(5)
    tipo registro domicilio
        carácter(*15) calle
        entero numero
        carácter(*20) ciudad
    fin
fin
Tipo ( alumno) a
carácter rta , esta
carácter(*5) lega
Inicio
abrir (22, “C:\informatica I \DATOS.DAT”, acción=”lectura”) !DATOS. DAT es el nombre
del archivo en disco
!22 es el número de referencia al archivo en el programa
rta ← x27;Sx27;
Repetir mientras( rta = x27;Sx27;) ! Este repetir mientras es para permitir hacer varias altas
    esta ← ‘N’
    escribir ( “Legajo : ” )
    leer ( lega )
    Repetir mientras (NO ( EOF(22) y esta = ‘N’)) ! busca si este registro ya está grabado
        leer ( 22 , a )
        Si ( a.legajo = lega) entonces
            escribir ( “ Ya está este Legajo – Alta rechazada” )
            esta ← x27;Sx27;
        fin si
    fin mientras
    Si (esta = ‘N’) entonces
        escribir ( “Apellido: ” ) ! lo vamos a grabar al final del archivo, para ello será
        !necesario cerrarlo como “lectura”
        leer ( a . apeynom . apellido ) ! abrirlo como “agregado”
        escribir ( “Nombre: ” )
        leer ( a . apeynom . nombre )
        escribir (“Ingrese las cinco notas”)
        Repetir para i ←1, 5
            leer ( a . notas ( i ) )
        fin para
        escribir (“Ingrese Calle Nro. Ciudad”)
        leer ( a . domicilio. calle , a . domicilio . numero , a . domicilio . ciudad )
        a . legajo ← lega
        cerrar (22)
        abrir (22,”C:\ informática I \ DATOS.DAT” , acción= “agregado”)
```

```
        escribir (22, a)
        cerrar (22)
        abrir (22,"C:\ informática I \DATOS.DAT" ,acción ="lectura")
    fin si
    escribir (" Quiere ingresar otro alumno? (Sx2F;N)")
    leer ( rta )
fin mientras
cerrar ( 22 )
FIN Altas
```

- BAJAS

Hacer un algoritmo que permita eliminar (BAJAS) registros del archivo DATOS.DAT, cuyos registros fueron
descriptos en el ejemplo anterior. Hacemos el algoritmo para una sola baja.

```
Algoritmo Bajas
Tipo registro alumno
    carácter(*5)legajo
        tipo registro apeynom
    carácter(*20) apellido
    carácter(*25) nombre
fin
    entero notas(5)
tipo registro domicilio
        carácter(*15) calle
        entero numero
        carácter(*20) ciudad
    fin
fin
Tipo ( alumno) a
carácter esta
carácter(*5) lega
Inicio
abrir (1,"C:\Informatica I \DATOS.DAT, acción ="lectura" )
! la idea es copiar uno a uno los registros de DATOS.DAT
! en AUXI.DAT con excepción del que queremos dar de baja
abrir ( 2 , "C:\Informatica I\AUXI.DAT,acción ="escritura" )
está ←'N' ! luego eliminamos DATOS.DAT y
escribir ( "Legajo a eliminar : " ) ! renombramos AUXI.DAT como DATOS.DAT
leer ( lega ) ! ahora busca si este legajo está en el
! archivo
Repetir mientras( NO ( EOF(1 )))
    leer ( 1 , a )
    Si ( lega = a.legajo) entonces
        escribir ( " Este Legajo corresponde a ", a . apeynom . apellido , " y se borrará")
        esta ← x27;Sx27; ! NO lo graba en AUXI.DAT
```

```
    sino
        escribir (2 , a ) ! graba el registro en AUXI.DAT
    fin si

fin mientras
Si (esta = 'N') entonces
    escribir ("No se encontró ese Legajo ", lega )
fin si
cerrar ( 1 )
cerrar ( 2 )
borrar (DATOS.DAT)
renombrar (AUXI.DAT como DATOS.DAT)
FIN Bajas
```

- MODIFICACIONES

Hacer un algoritmo que permita modificar (MODIFICACIONES) registros del archivo DATOS.DAT, cuyos registros fueron descriptos en el ejemplo anterior. Hacemos el algoritmo para una sola modificación.

```
Algoritmo Modificar
Tipo registro alumno
    carácter(*5)legajo
    tipo registro apeynom
        carácter(*20) apellido
        carácter(*25) nombre
    fin
    entero notas(5)
    tipo registro domicilio
        carácter(*15) calle
        entero numero
        carácter(*20) ciudad
    fin
fin
Tipo (alumno) a
carácter esta
entero i
cadena(*5) lega
Inicio
abrir ( 11 , "C:\Informatica I \DATOS.DAT", acción ="lectura" )
abrir ( 12 , "C:\Informatica I \AUX.DAT", acción= "escritura" )
esta ← 'N';
escribir ( "Legajo : " )
leer ( lega )
Repetir mientras ( NO( EOF ( 11 ) )) ! ahora busca si este registro ya está grabado

    leer ( 11 , a )
```

```
    Si ( a . legajo = lega) y ( esta =‘N’) entonces

        Llamar mostrar ( a )
        Llamar cambios ( a )

        escribir (12 , a)

        esta ← x27;Sx27;
    sino


        escribir (12 , a)
    fin si

fin mientras
Si( esta = ‘N’) entonces
    escribir ( “ El Legajo ”, lega ,"no existe")
fin si
cerrar ( 11 )
cerrar ( 12 )
borrar (DATOS.DAT)
renombrar (AUX.DAT como DATOS.DAT)
```
FIN Modificar

Subrutina mostrar (a)
Tipo(alumno) a
Inicio
```
escribir ( “Legajo : ” , a . legajo)
escribir ( “Apellido : ”, a . apeynom . apellido)
escribir ( “Nombre : ”, a . apeynom . nombre)
escribir ( “Calle : ”, a . domicilio . calle , a . domicilio . numero)
escribir ( “Localidad : ” , a . domicilio . ciudad)
escribir ( “Notas :”)
Repetir para i ← 1, 5
    Escribir ( a . notas[ i ] )
fin para
```
FIN mostrar

Subrutina cambios (a)
Tipo(alumno) a
Inicio
```
escribir (“Ingrese, en el mismo orden todos los datos correctos”)
leer (a . legajo)
leer (a . apeynom . apellido)
leer (a . apeynom . nombre)
```

leer (a . domicilio . calle , a . domicilio . numero, a.domicilio.ciudad)
Repetir para i ←1, 5
    leer (a . notas (i))
fin para
FIN cambios

b.1) Procesamiento de archivos con Organización Directa

De acuerdo a lo explicado en el item b), desarrollaremos un ejemplo.

Ejemplo:
En un archivo directo **MATRIZ.DAT** se han grabado los elementos de una matriz cuadrada real de 100 filas por 100 columnas, en orden de filas. Cada elemento ocupa un registro. Sin necesidad de cargar la matriz en memoria:
a) Calcular el promedio de los elementos de la primera columna de la matriz.
b) Reemplazar cada elemento de la última fila por dicho promedio.

Algoritmo directos
tipo registro reg1
        real: x
    fin_registro
    entero: i
    real: suma, prom
tipo(reg1) elem
Inicio
abrir (20,"MATRIZ.DAT",acceso= directo, longitud registro reg1)
! se especifica no sólo la referencia 20 y el nombre del archivo, sino también que es de acceso ! directo yla longitud del tipo de registro, en este caso reg1, que por contener un solo campo ! real ocupará 4 bytes.
suma ← 0
Repetir para i ← 1 , 9901, 100
    ! localizar cada elemento de la primera columna de la matriz; el
    ! primero está en el registro 1, el segundo en el 101, el tercero en
    ! el 201, hasta el centésimo en el 9901.
    leer (20, registro i: elem) **! leer el registro que está en la posición i (completo).**
    suma ← suma + elem.x **!acumular en suma el valor del único campo del registro**
finpara
prom ← sumax2F;100 **! promedio calculado**
elem.x ← prom **! asignarlo para modificar la última fila**
Repetir para i ← 9901, 100000 ! **los elementos de la última fila ocupan esas posiciones**
    escribir (20, registro i: elem) **! escribir el registro completo en la posición dada**
finpara
cerrar (20)
FIN

9 — Representación de la Información en una Computadora

Pablo Augusto Magé I.

Un bit no tiene color, ni tamaño, ni peso y puede desplazarse a la velocidad de la luz. Es el elemento atómico más pequeño en la cadena de ADN de la información...

Nicholas Negroponte (Ser digital 1995)

9.0.1 Introducción

En una computadora se maneja información o datos. El dato se convierte en el elemento esencial sobre el cual opera una computadora.
En la solución de un problema es importante que el diseño del algoritmo de un problema como el diseño de los datos que se manejan en el problema quede bien estructurado.Dependiendo del tipo o tipos de datos que se manejen, éstos se clasifican en:
Datos simples: Son aquellos datos que manejan un valor de un solo tipo.Datos compuestos: Son aquellos datos que manejan valores de distintos tipos.
En este capítulo se realizará una descripción de como se representan los datos en una computadora independiente de que sean datos simples o datos compuestos

9.1 Sistemas de Numeración. Representación Interna de la Información

Cristina I. Alarcón

TIPOS DE DATOS ESTANDAR

Para poder hacer un uso eficiente de los tipos de datos estándar y las variables perteneciente a ellos, **entero**, **carácter** y **real en simple precisión**, es útil conocer la representación interna de sus valores, es decir como realmente los reconoce" la computadora.Con tal fin haremos primeramente un repaso de los sistemas de numeración decimal, binario y hexadecimal.

SISTEMAS DE NUMERACION

DEFINICION: sistema de signos o símbolos utilizados para expresar los números

BASE: es la cantidad TOTAL de símbolos ó guarismos necesarios para representar los infinitos posibles números en un sistema**SIMBOLOS, DIGITOS, CIFRAS Ó GUARISMOS:** cada uno de los signos que expresan una cantidad

a) Sistema Decimal

BASE: 10

SIMBOLOS, DIGITOS, CIFRAS Ó GUARISMOS: 0, 1, 2, 3, 4, 5, 6, 7, 8,9

Tanto el sistema decimal como el binario y el hexadecimal son sistemas de representación numérica "posicional", porque cada dígito ó cifra contribuye al valor del número de acuerdo a su valor y posición dentro del mismo. Sólo es posible utilizar la notación posicional si existe un símbolo para el cero. El guarismo 0 permite distinguir entre 11, 101 y 1.001 sin tener que utilizar símbolos adicionales.En el sistema decimal todos los números se pueden expresar utilizando sólo diez guarismos, del 1 al 9 más el 0. La notación posicional simplifica todos los tipos de cálculo numérico por escrito.
El concepto de valores posicionales, que es fundamental para expresar los números enteros mediante la notación arábiga en una base de numeración cualquiera, puede extenderse para incluir los números fraccionarios. Si la base de numeración es 10, las distintas cifras de un número entero positivo "valen" su valor multiplicado por una potencia positiva de 10.Ejemplo: 2578 = 2 x 10 elevado a 3 + 5 x 10 elevado a 2 + 7 x 10 elevado a 1 + 8 x 10 elevado a 0En los números con partes fraccionarias, las cifras que forman dicha parte "valen" sus valores multiplicados por potencias negativas de 10. Estas cifras se sitúan a la derecha de la de las unidades, separadas de ésta por una coma.Las unidades fraccionarias a la derecha de la coma se llaman décimas, centésimas, milésimas, diezmilésimas,. . . , millonésimas.Ejemplo:2578,25 = 2 x 10 elevado a 3 + 5 x 10 elevado a 2 + 7 x 10 elevado a 1 + 8 x 10 elevado a 0 + 2 x 10 elevado a -1 + 5 x 10 elevado a -2

b) Sistema Binario

BASE: 2

SIMBOLOS, DIGITOS, CIFRAS Ó GUARISMOS: 0, 1

En este sistema cualquier número se representa por medio de los símbolos 0 y 1.
Los primeros 10 números en el sistema en base 2 y su correspondiente número decimal son:

Sistema Decimal	Sistema Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

De igual forma que en el Sistema Decimal, podemos representar un número cualquiera como sumas de potencias de la base 2
El número 1001 representa:1001 = 1 x 2 elevado a 0 + 0 x 2 elevado a 1 + 0 x 2 elevado a 2 + 1 x 2 elevado a 3

1001 = 1 + 0 + 0 + 8
1001 en base 2 = 9 en base 10
Para realizar las operaciones aritméticas con números en base 2 las reglas principales son:
1 + 1 = 10
$1 \times 1 = 1$
$1 \times 0 = 0$
1 + 0 = 1
Las operaciones de suma, resta y multiplicación se realizan de manera semejante a las del sistema decimal.

c) Sistema Hexadecimal

BASE: 16

SIMBOLOS, DIGITOS, CIFRAS Ó GUARISMOS: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

En este sistema cualquier número se representa por medio de los dígitos 0,..,9 y las letras A,.., F.
Los primeros15 números en los tres sistemas son:

Sistema Hexadecimal	Sistema Binario	Sistema Decimal
0	0000 0	
1	0001 1	
2	0010 2	
3	0011 3	
4	0100 4	
5	0101 5	
6	0110 6	
7	0111 7	
8	1000 8	
9	1001 9	
A	1010 10	
B	1011 11	
C	1100 12	
D	1101 13	
E	1110 14	
F	1111 15	

d) Conversión

1. DE DECIMAL A BINARIO

Para convertir un número n dado en base 10 a base 2, se divide, en el sistema decimal, n por 2, el cociente se divide de nuevo por 2 y así sucesivamente hasta que se obtenga un cociente cero. Los restos sucesivos de esta serie de divisiones son los dígitos que expresan a n en base 2. La base se suele escribir como un subíndice del número.
Ejemplo:
75 en base 10 lo convertimos a base 2 de la siguiente manera:

nro	resto
75	1
37	1
18	0
9	1
4	0
2	0
1	

75 en base 10 = 1001011 en base 2

2. DE BINARIO A DECIMAL

Se procede a realizar la sumatoria de los dígitos binarios (bits) desde el dígito binario menos significativo (el primero empezando por la derecha) MULTIPLICADOS POR LAS POTENCIAS DE 2, COMENZANDO POR EL CERO.Si aplicamos esto al número binario recién calculado:
1 0 0 1 0 1 1 en base 2 = 1 x 2 elevado a 0 + 1 x 2 elevado a 1 + 0 x 2 elevado a 2 + 1 x 2 elevado a 3 + 0 x 2 elevado a 4 + 0 x 2 elevado a 5 + 1 x 2 elevado a 6
= 1 + 2 + 0 + 8 + 0 + 0 + 64
= 75 en base 10

3. DE DECIMAL A HEXADECIMAL

El número 75 en base 10 lo convertimos a base 16 de la siguiente manera:

nro	resto
75	11 = B
4	

75 en base 10 = 4B en base 16

4. DE HEXADECIMAL A DECIMAL

Se procede a realizar la sumatoria de los dígitos binarios (bits) desde el dígito binario menos significativo (el primero empezando por la derecha) MULTIPLICADOS POR LAS POTENCIAS DE 16, COMENZANDO POR EL CERO.Si aplicamos esto al número hexadecimal recién calculado:
4B en base 16 = 11 x 16 elevado a 0 + 4 x 16 elevado a 1
= 11 + 64
= 75 en base 10

5. DE HEXADECIMAL A BINARIO

Se toma cada dígito ó símbolo hexadecimal y se lo representa por el correspondiente binario. Dado que 2 elevado a 4 = 16, 4 dígitos binarios corresponden a un hexadecimal.

4B en base 16 = 0100 1011 en base 2

6.DE BINARIO A HEXADECIMAL

De derecha a izquierda se van tomando cada 4 dígitos binarios y se los va remplazando por el correspondiente hexadecimal

0100	1011
4	B

NOTA: En la parte correspondiente a representación interna de reales en simple precisión se verá sobre ejemplos la conversión entre sistemas numéricos con números con parte fraccionaria.

REPRESENTACION INTERNA DE LA INFORMACION

Un número en sistema binario (y por lo tanto su correspondiente en el sistema decimal) se puede representar internamente en una computadora, por ejemplo, con las distintas posiciones de una serie de interruptores. Si adoptamos para la posición de .encendido.al valor 1, y .apagado.al 0, observamos que son necesarios dos estados, fundamentalmente. Esta consideración ha llevado a los diseñadores de computadoras a elegir este sistema como base para la representación de información, en consecuencia la información que almacena una computadora está codificada en el sistema binario.

a) REPRESENTACION INTERNA DE CARACTERES

En nuestra representación interna vamos a utilizar 1 byte = 8 bits (dígitos binarios)

Cantidad de combinaciones posibles : 2 elevado a 8 = 256, esto quiere decir que se pueden representar 256 caracteres diferentes.Existen varios códigos que establecen una correspondencia entre número y carácter, uno de ellos es el ASCII (Código Americano Estándar para Intercambio de Información)Casi todos los códigos normalizan menos de 256 caracteres, dejando algunos disponibles para el usuario.

ASCII

Nro.	Carácter
65	A
66	B
67	C
....	
90	Z
48	0
49	1
......	
57	9
a	97
b	98
c	99
....	
z	122

Por lo tanto cuando queremos obtener la representación interna de un carácter debemos:

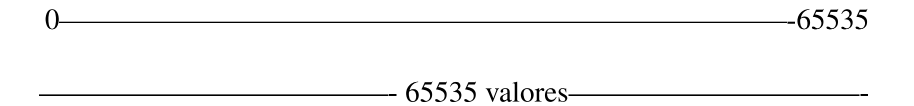
1. Conocer el ASCII correspondiente
2. Convertirlo al sistema de numeración binaria
3. Expresarlo con 8 bits

Ejemplo:
Obtener la representación interna de la letra C
1-El código ASCII correspondiente es 67

2- La conversión de 67 a sistema binario es =
67 1
33 1
16 0
8 0
4 0
2 0
1
3- Representación:
0100 0011

b) REPRESENTACION INTERNA DE ENTEROS

Los enteros los vamos a representar en 2 bytes o sea 16 bits, pero como vamos a representar enteros signados (con signo: positivos o negativos) el bit más significativo, .el de más a la izquierda” tendrá valor 0 (cero) si el entero representado fuera positivo y tendrá valor 1 si el entero representado fuera negativo.Cantidad máxima de enteros a representar: 216 = 65536



Pero debemos recordar que tenemos que representar los enteros + y - , por lo tanto al intervalo [0, 65535] lo desplazamos 32768 valores hacia la izquierda y tenemos el siguiente intervalo para la representación de los positivos y negativos

Máximo entero + = 65535-32768= 32767
Cero = 32768 – 32768 = 0
Menor entero negativo = 0 -32768 = - 32768

0 - 32768	32768-32768	65535 -32768
-32768	0	32767

Vemos que para la representación de enteros signados en dos bytes el intervalo será: [- 32768, 32767].

Deducimos que el intervalo de representación o rango para enteros signados se puede generalizar como:(*) **[- 2 elevado a (n-1) , (2 elevado a (n-1)) -1]**, donde n es la cantidad de bits utilizados en la representación.

1. Representación de enteros positivos

1- Convertir el entero (valor absoluto) a binario.2- Representar al mismo con 16 bits.3- El bit más significativo, el bit de signo, (el de más a la izquierda), por el convenio explicado antes valdrá cero.
Ejemplo:Representación de +123

1. 1 23 en base 2 = 1111011
2. 0000 0000 0111 1011
3. 0 000 0000 0111 1011

Bit de signo

2. Representación de enteros negativos

En la representación de enteros negativos se utiliza el **complemento a la base 2**, y esto se debe a los fines de facilitar las operaciones posteriores que pudieran realizarse.

Recordamos una de las reglas para obtener la notación de complemento a 2.
"De derecha a izquierda copiamos el número binario exactamente igual hasta el primer 1 al cual también lo copiamos y a partir de aquí invertimos los valores, donde existe 1 colocamos 0 y viceversa, con la excepción del bit de signo"Ejemplo: 1 000 1000 1010 1000
complemento a 2 1 111 0111 0101 1000

El complemento a la base sólo lo vamos a utilizar para la representación de enteros negativos

Para obtener la representación interna de enteros negativos debemos:

1. Convertir el entero (valor absoluto) a binario
2. Representarlo con 16 bits
3. Complementar a 2
4. El bit de signo por convenio valdrá 1

Ejemplo:
Representación de -123

1. 123 en base 2 = 1111011
2. 0000 0000 0111 1011
3. 0111 1111 1000 0101
4. 1 111 1111 1000 0101

Bit de signo

c) REPRESENTACION INTERNA DE REALES EN SIMPLE PRECISION
Dado que el conjunto de los números reales es infinito, la cantidad de los mismos, como así también las çifras significativas" que tuvieran dichos números dependerá de la cantidad de bytes que se otorga para su representación. Nosotros le otorgamos 4 bytes
Adoptamos como forma de representación la exponencial que constituye la notación en punto flotante o también conocida como notación científica.
En esta forma de representar un número, el mismo puede asumir infinitas representaciones, es

como que la coma puede "flotar” k lugares a la derecha o k lugares a la izquierda del mismo. Por ejemplo:

En base 10
234,4567 = 23,44567 * 10 elevado a 1 = 23244567 *10 elevado a -5 = 2,344567 * 10 elevado a 2 = 0,2344567 * 10 elevado a 3

En base hexadecimal
23,DEF = 23DEF * 16 elevado a -3 = 2,3DEF * 16 elevado a 1 = 0,23DEF *16 elevado a 2 = 0,00023DEF * 16 elevado a 5

Observamos que el número queda representado, cualquiera sea el sistema de numeración posicional, como el producto de dos factores:

Mantisa * Base del sistema elevado a Exponente

Donde Mantisa es el número en sí, en el cual la parte entera del mismo puede tener una cantidad de dígitos que puede variar entre la cantidad total de dígitos que tiene el número y cero dígito como parte entera.Si nos remitimos al primer ejemplo, la mantisa en cada una de las distintas representaciones del 234,4567 son:
23,44567 - 23244567 - 2,344567 - 0,2344567

Normalización
Se toma como estándar la representación denominada NORMALIZADA, que consiste en que la mantisa no tiene parte entera y el primer dígito a la derecha del punto es significativo, es decir es distinto de cero, salvo en la representación del número cero.**Mantisa**: es un número menor que la unidad, es decir parte entera cero seguida de un primer dígito significativo, o sea distinto de cero.**Exponente**: es el número al que hay que elevar la base del sistema, indicando este valor cuántos lugares debe trasladarse la coma.

Ejemplo		mantisa	base	exponente
Sistema decimal:	0,0000123	= 0,123	x 10	elevado a -4
Sistema hexadecimal:	4B	= 0.4B	x 16	elevado a 2

Dado que con el sistema hexadecimal la información se representa más .empaquetada”, pues un dígito hexadecimal es equivalente a cuatro dígitos binarios, la mayoría de los sistemas de computación almacenan primeramente la información en hexadecimal y luego la convierten a binario aprovechando la directa conversión de un sistema a otro.Una de las formas más frecuente de disposición de los 4 bytes es la siguiente:

exponente	mantisa		
1er byte	2do byte	3er byte	4to byte

Bit del signo: 0 para positivo, 1 para negativo
En los 7 restantes bits del primer byte se representa el exponente
Del segundo byte al cuarto se representa la mantisa

Ejemplo:
Representación de 123,45 en 4 bytes
1- Convertimos el número decimal a hexadecimal, separando en la conversión la parte entera de la parte decimal:
123 base 10 = 7 B base 16
2- Convertimos la parte fraccionaria a hexadecimal:

2-1-
0,45
x 16
———
270
+ 45
———
7,20
Primer cifra fraccionaria en hexadecimal = 7

2-2- Del valor obtenido tomamos 0,20 y nuevamente volvemos a multiplicar por 16, y así sucesivamente hasta obtener un valor periódico o cero.
0,20
x 16
———-
3, 20
Segunda cifra fraccionaria hexadecimal = 3

2-3- Realizamos el mismo procedimiento que en el punto 2

0,20
x 16
———-

3,20
Tercer cifra fraccionaria hexadecimal = 3

123,45 en base 16 = 7B, 7333 periódico, dando en su representación un error de truncamiento.

3- Normalizamos, expresamos el número en notación exponencial con mantisa de parte entera cero y el primer dígito después de la coma debe ser distinto de cero

7B, 7333 = 0,7B7333 x 16 elevado a 2

4- Mantisa = 0,7B7333, la cual la vamos a representar del segundo byte al cuarto byte, todas las cifras significativas, es decir 7B7333, convirtiendo cada una al binario correspondiente.

5- Representación de la característica del exponente: como dijimos antes, lo vamos a hacer en 7 bits del primer byte, esto significa que podemos representar tantos valores del exponente como: 27 = 128 valores de los cuales tendremos en cuenta la representación de los valores + y - del exponente, por lo que desplazamos la escala 64 lugares hacia la izquierda. Es por esto que decimos que en los 7 bits que le siguen al bit de signo del primer byte representamos la CARACTERISTICA DEL EXPONENTE.
Reemplazando en la formula de más arriba (*) (en el ítem b-Representación interna de enteros), el rango de representación, no del exponente sino de la CARACTERISTICA DEL EXPONENTE será:

elevado a (7-1), (2 elevado a (7-1)) -1
= [-64, 63] en base 10 = [-40, 3F] en base 16

En nuestro caso el exponente es 2 al que sumaremos 40 en base 16 (es 64 en base 10) debido al desplazamiento explicado y recordando que las reglas que rigen las operaciones de suma y resta en el sistema hexadecimal son las mismas que en el sistema decimal.CARACTERISTICA DEL EXPONENTE = 40 en base 16 + EXPONENTE en base 16
40 + 2 =42 en base 16

Por lo tanto la representación interna de 123,45 = 7B, 7333 = 0,7B7333 x 16 elevado a 2 será:

01000010	0111	1011	0111	0011	0011	0011
4 2	7	B	7	3	3	3
Bit de signo	42	7B7333				

Nota: si el número a representar no tuviera fracción periódica, los bits menos significativos (los bits de más a la derecha) se completan con ceros en la representación de la mantisa.

Ejemplos:
a) Hallar la representación interna de 1,5
1- Convertir la parte entera a hexadecimal 1 en bas 10 = 1 en base 16
2- Convertir la parte fraccionaria a hexadecimal
0,5
x 16
———
8, 00

Primer y único dígito fraccionario hexadecimal 8, dado que la fracción 00 nos daría cero si la multiplicamos x 16.
1,5 en base 10 = 1,8 en base 16

3-Normalizamos
1,8 = 0,18 x 16 en base 1

4- Mantisa = 0,18

5-Característica del Exponente: = 1 en base 16 + 40 en base 16 = 41 en base 16

1000001	0001	1000	0000	0000	0000	0000
4 1	1	8	0	0	0	0

b) Hallar el número real representando en los siguientes 32 bits.

1 010 1011 0001 0010 0011 0000 1111 0000,
convirtiendo de binario a hexadecimal:
- 2 B 1 2 3 0 F 0

El bit de signo es 1 por lo que el número representado es negativo

CARACTERISTICA DEL EXPONENTE = 2B en base 16

MANTISA = 0.1230F en base 16

CARACTERISTICA DEL EXPONENTE = 40 en base 16 + EXPONENTE, de donde

EXPONENTE = CARACTERISTICA DEL EXPONENTE - 40 en base 16, reemplazando:
EXPONENTE = 2B - 40 en base 16

El número hexadecimal 2B es más chico que el hexadecimal 40, recordando que las reglas que rigen la resta en el hexadecimal son las mismas que en el sistema decimal podemos expresar: 2B – 40 = - (40 +- 2B), resolviendo el segundo miembro de esta igualdad:

4 queda en 3 4 0 (10) {cuánto le falta a B para llegar a la base 10 (=16 en decimal):
- 2 B (B) C, D, E, F, 10 = 5}
EXPONENTE - 1 5

El número en hexadecimal en notación científica normalizada es:

Número en base dieciseis = - 0,1230F

Si queremos expresar el número en el sistema decimal

(1*16 elevado a -1 + 2*16 elevado a -2 + 3*16 elevado a -3 + 15*16 elevado a -5) * 16 elevado a -21 = número en base 10

(0,0625 + 0,0078125 + 0,00073242187 + 0,0000143051) * 16 elevado a -21 = número en base 10

0.071059226 * 16 elevado a -21 = número en base 10

Aplicando logaritmo a ambos miembros de esta igualdad

log 0,071059226 + (-21)* log 16 = log número

-1,1483795 - 25,28652 = log número

-26,4348995 = log número

Número en base 10 = antilogaritmo (- 26,4348995)

Número en base 10 = -3,673673 * 10 elevado a -27

c) Hallar el rango de representación de los reales en simple precisión, en 4 bytes

Valor máximo representado en valor absoluto = 0,FFFFFF * 16 elevado a 3F

0,FFFFFF = 15/16 + 15/256 + 15/4096 + 15/65536 + 15/16 elevado a 5 + 15/16 elevado a 6 = ~1

Valor mínimo representado en valor absoluto = 0,FFFFFF * 16 elevado a -40

Por lo tanto el rango de representación en 4 bytes en hexadecimal es:

$\pm$ [16 elevado a -40, 16 elevado a 3F]

Si queremos expresar dicho intervalo en el sistema decimal tendremos:

16 elevado a -40 =16 elevado a -64 = 10 elevado a x

aplicando logaritmo en base 10 a ambos miembros:

-64 * log 16 = x * log10 donde log10 =1

despejando X :

X= -64 * log 16 = ~- 78
Procediendo de la misma forma

16 elevado a 3F = 16 elevado a 63 = ~10 elevado a 75

donde el intervalo expresado en sistema decimal es:

$\pm$[10 elevado a -78 , 10 elevado a 75]

9.2 Sistemas de numeración para la representación a Bajo y Alto Nivel

Pablo Augusto Magé I.

9.2.1 Representación a Bajo Nivel

La representación de las información o los datos a bajo nivel, también es conocida como representación a nivel de máquina. En este tipo de representación de la información, los datos en una computadora se representan como una secuencia de bits (contracción de binary digit cuyos valores son los dígitos 0 y 1).Para representar la información numérica se utilizan los sistemas numéricos. A continuación se hace una breve descripción de los Sistemas numéricos más utilizados, centrándonos en el Sistema Binario por ser la representación base para otros sistemas numéricos.

Sistemas numéricos

Existen diferentes sistemas numéricos donde los más utilizados son los siguientes:

- Sistema Decimal Se utiliza como base 10, posee 10 símbolos para representar la información numérica, esto es: 0,1,2,3,4,5,6,7,8 y 9.

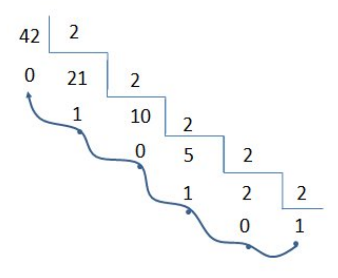
- Sistema Octal La base utilizada es el 8, posee 8 símbolos para representar las cantidades numéricas: 0,1,2,3,4,5,6 y 7.

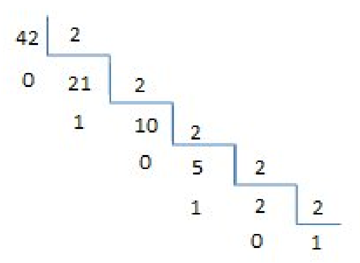
- Sistema Hexadecimal La base utilizada es el 16, posee 16 símbolos para representar las cantidades numércias: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E y F.

- Sistema binario En los componentes internos que utilizan las computadoras se utiliza el sistema binario. Se denomina así porque se utiliza como base el 2, se emplean 2 símbolos para representar las cantidades numéricas: 0 y 1. Estos valores son conocidos como bits (contracción de binary digit).

Representación de un número decimal en binario

Para obtener la representación de un número decimal en binario se debe dividir el número decimal por la base del sistema binario, es decir 2.Por ejemplo: Convertir el número 42 (en base 10) al sistema binario:





Para obtener el número equivalente en sistema binario se toman los residuos de cada división comenzando desde la derecha hacia la izquierda, por tanto el resultado es:

42(base 10) = 101010(base 2)

Representación de un número binario en sistema decimal

Para obtener la representación de un número binario a su equivalente en sistema decimal se multiplica el dígito binario por el peso de dicho número que ocupa en el número binario, el peso por 2 elevado a la p, donde p es la posición del dígito binario. El menor peso está ubicado en la última posición del lado derecho.Por ejemplo:101010 en binario equivale a:

$$101010_2 = 0x2^0 + 1x2^1 + 0x2^2 + 1x2^3 + 0x2^4 + 1x2^5$$
$$= 0 + 2 + 0 + 8 + 0 + 32$$
$$= 42_{10}$$

Operaciones con números binarios

Suma de números binarios

Los resultados de las combinaciones de sumas entre los dígitos binarios 0 y 1, se resumen en la Tabla 1

0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	10 (Equivalente a 2 en sistema decimal)

Tabla 1: Casos básicos de la operación suma

Por ejemplo:
Realizar la siguiente suma de binarios 101010 + 101001

```
   1 0
  1 0 1 0 1 0
  1 0 1 0 0 1 +
 --------------
 1 0 1 0 0 1 1
```

Resta de números binarios

Los casos básicos de la operación resta entre los dígitos binarios 0 y 1 se resumen en la Tabla 2

0 - 0	0
0 - 1	1(Ver nota)
1 - 0	1
1 - 1	0

Tabla 2: Casos básicos de la operación resta

Nota: En el caso: 0 – 1, se debe sustraer una unidad de la cifra que está a la izquierda del dígito binario para realizar esta operación, sino existe un dígito esta operación no puede ser realizada.Por ejemplo:
Realizar la siguiente resta de los números binarios 111010 - 101100

```
 01
   0 1
  111010
 -101100
 -------
  001110
```

Multiplicación de números binarios

Los casos básicos de la operación multiplicación entre los dígitos binarios 0 y 1 se resumen en la Tabla 3.

0 x 0	0
0 x 1	0
1 x 0	0
1 x 1	1

Tabla 3: Casos básicos de multiplicación de números binarios

Por ejemplo:
Realizar la multiplicación de los números binarios 101010 x 1001

```
     101010
   x   1001
   --------
     101010
    000000
   000000
  101010
  ---------
  101111010
```

División de números binarios

Se sigue el mismo procedimiento de una división entre números decimales. Se intenta dividir el dividendo por el divisor tomando el mismo número de cifras en ambos lados, sino se puede dividir se toma una cifra más del dividendo.Por ejemplo:
Realizar la división de los números binarios

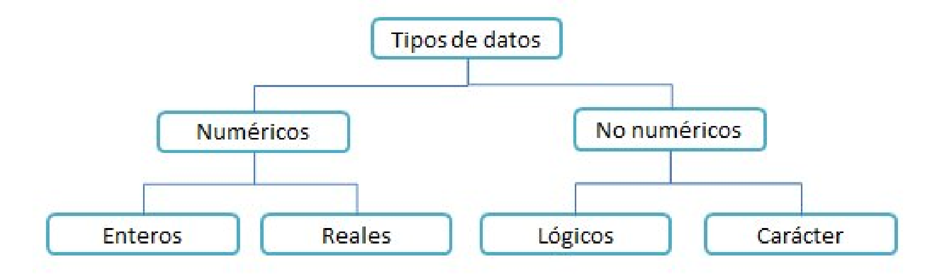
```
 1011011 | 101
-101     |------
-------    1001
 000101
    -101
    ----
    0001
```

9.2.2 Representación a Alto Nivel

Se utilizan abstracciones que permiten ignorar la representación a bajo nivel o a nivel de máquina de los datos, estas abstracciones originan el concepto de tipos de datos.

Tipos de datos

Los tipos de datos simples se clasifican en datos numéricos y no numéricos. La clasificación completa de estos tipos de datos se muestra en la siguiente figura:



a- Datos Numéricos

Este tipo de dato maneja los valores numéricos que se pueden agrupar en 2 formas:

- Tipo entero
- Tipo real

Tipo entero: Corresponde a un subconjunto finito de los números enteros. Los números enteros no poseen la parte decimal y pueden ser positivos o negativos.
En la Tabla 4 se describen las cantidades de enteros que se pueden representar dependiendo de la arquitectura manejada.

Arquitectura	Rango - con signo	Rango - sin signo
16 bits	-32768 a + 32767	0 a 65.535
32 bits	-2.147.483.648 a +2.147.483.647	0 a 4.294.967.295
64 bits	-9,223,372,036,854,775,808 a +9,223,372,036,854,775,807	0 a 18,446,744,073,709,551,615

Tipo real: Corresponde a un subconjunto de los números reales. Los números reales poseen una parte decimal y pueden ser positivos o negativos.En la Tabla 5 se describen las cantidades de reales que se pueden representar según el estándar IEEE 754 (2008).

Arquitectura	Rango aproximado	Precisión
16 bits	??	3??
32 bits	±1,5e-45 a ±3,4e38	6 ½
64 bits	±5,0e-324 a ±1,7e308	15

b- Datos No Numéricos

Este tipo de dato maneja los valores no numéricos que se pueden agrupar en 2 formas:

- Tipo Lógico
- Tipo Carácter

c- Datos Lógicos

Este tipo de dato maneja solo dos valores: Falso(F) o Verdadero(V). Este tipo de datos es utilizado para definir decisiones en un problema, mediante la definición de condiciones.

d- Datos Carácter

Los datos carácter están conformados por el conjunto ordenado de caracteres que la computadora puede representar. En general los caracteres que se pueden representar en una computadora se pueden clasificar en las siguientes categorías:Caracteres alfabéticos: Por ejemplo: A,a,B,b,C,c

Caracteres numéricos: Por ejemplo: 1,2,3 etc.

Caracteres especiales: Por ejemplo: +.-,*,/ etc.

Codificación de caracteres
Este proceso consiste en asignarle a cada carácter de un alfabeto un valor numérico usando una norma de codificación, a este valor numérico se le denomina punto de codificación. Los puntos de codificaciónse representan por uno o más bytes.

Normas de codificación
Las normas de codificación definen la forma como un carácter se codifica en un punto de codificación. Ejemplos de normas de codificación son: ASCII, ASCII Extendido, Unicode

ASCII
Esta norma utiliza 1 byte para codificar caracteres, usando 7 bits para codificar el carácter y 1 bit para detectar errores de paridad, por lo tanto sólo se pueden codificar 128 caracteres. Entre los símbolos codificados en la Norma ASCII se incluyen mayúsculas y minúsculas del abecedario inglés, además de cifras, puntuación y algunos “caracteres de control”, pero el ASCII no incluye ni los caracteres acentuados ni el comienzo de interrogación que se usa en castellano, ni tantos otros símbolos (matemáticos, letras griegas,...) que son necesarios en muchos contextos.

ASCII Extendido
Con el fin de superar las deficiencias de la Norma ASCII se definieron varios códigos de caracteres de 8 bits, entre ellos el ASCII extendido. Sin embargo, el problema de estos códigos de 8 bits es que cada uno de ellos se define para un conjunto de lenguas con escrituras semejantes y por tanto no dan una solución unificada a la codificación de todas las lenguas del mundo. Es decir, no son suficientes 8 bits para codificar todos los alfabetos y escrituras del mundo.

Unicode
La Norma Unicode emplea 2 bytes (16 bits) para representar cada carácter. Esta norma permite la representación de cualquier carácter, en cualquier lenguaje, permitiendo abarcar todos los alfabetos europeos, ideogramas chinos, japoneses, coreanos, muchas otras formas de escritura, y más de un millar de símbolos locales.

10 — Bibliografía

Brookshear J.Glenn. (1995) - Introducción a las Ciencias de la Computación. 4ta. Edición Addison-Wesley

Joyanes Aguilar L. - Fundamentos de Programación. Algoritmos, estructuras de datos y objetos (2009) 3ra Edición McGraw-Hill.

Lage F.; Cataldi Z.,;Salgueiro F., (2008) - Fundamentos de Algoritmos y Programación,Editorial NUEVA LIBRERÍA

Braunstein S.; Gioia A. (1995) - Introducción a la programación y a las estructuras de datos. Editorial EUDEBA, BuenosAires Argentina

Kernighan Brian W.; Ritchie Dennis M. - El Lenguaje de Programación C. 2da. Edición Prentice-Hall Hispanoamericana

Marzal Andrés; Gracia Isabel - Introducción a la programación con Python - 2003

Wirth Niklaus (1985) -Algoritmos + Estructuras de Datos = Programas Ediciones. Ed. del Castillo Madrid

Kerrigan Jim (1993) -Migrating to FORTRAN 90. 5ta. Edición O'Reilly & Associates, Inc

Knuth Donald - El arte de programar ordenadores. Vol. 1 . Algoritmos fundamentales. Ed. Reverté

Tucker A.;Cupper R.; Bradley W.J.; Garnick D.- Fundamentos de Informática, lógica, resolución de problemas, programas y computadoras, Mc
Graw Hill, 1994 J.

Taiana Aída; Moreli María Alicia; Mainieri Rosanna; Alarcón Cristina - Reflexiones en la Búsqueda de una Didáctica Específica de la
Algoritmia para la Programación (2011), Editorial de la Universidad Nacional de Rosario.



Edición: Marzo de 2014.
Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.

